

Improving the betweenness centrality of a node by adding links

ELISABETTA BERGAMINI, Karlsruhe Institute of Technology (KIT)

PIERLUIGI CRESCENZI, University of Florence

GIANLORENZO D'ANGELO, Gran Sasso Science Institute (GSSI)

HENNING MEYERHENKE, Karlsruhe Institute of Technology (KIT)

LORENZO SEVERINI, Gran Sasso Science Institute (GSSI)

YLLKA VELAJ, Gran Sasso Science Institute (GSSI)

Betweenness is a well-known centrality measure that ranks the nodes according to their participation in the shortest paths of a network. In several scenarios, having a high betweenness can have a positive impact on the node itself. Hence, in this paper we consider the problem of determining how much a vertex can increase its centrality by creating a limited amount of new edges incident to it. In particular, we study the problem of maximizing the betweenness score of a given node – Maximum Betweenness Improvement (MBI) – and that of maximizing the ranking of a given node – Maximum Ranking Improvement (MRI). We show that MBI cannot be approximated in polynomial-time within a factor $(1 - \frac{1}{2e})$ and that MRI does not admit any polynomial-time constant factor approximation algorithm, both unless $P = NP$. We then propose a simple greedy approximation algorithm for MBI with an almost tight approximation ratio and we test its performance on several real-world networks. We experimentally show that our algorithm highly increases both the betweenness score and the ranking of a given node and that it outperforms several competitive baselines. To speed up the computation of our greedy algorithm, we also propose a new dynamic algorithm for updating the betweenness of one node after an edge insertion, which might be of independent interest. Using the dynamic algorithm, we are now able to compute an approximation of MBI on networks with up to 10^5 edges in most cases in a matter of seconds or a few minutes.

CCS Concepts: **•Mathematics of computing** → **Approximation algorithms**; **•Theory of computation** → *Dynamic graph algorithms*; **•Applied computing** → *Sociology*;

Additional Key Words and Phrases: Betweenness Centrality, Graph Augmentation, Greedy Algorithm, Network Analysis

1. INTRODUCTION

In recent years, the analysis of complex networks has become an extremely active research area. One of the main tasks in network analysis is the ranking of nodes based on their structural importance. Since the notion of importance can vary significantly depending on the application, several *centrality measures* have been introduced in the literature. One of the most popular measures is *betweenness centrality*, which ranks the nodes according to their participation in the shortest paths between other node pairs. Intuitively, betweenness measures a node’s influence on the flow circulating through the network, under the assumption that the flow follows shortest paths.

Computing betweenness centrality in unweighted graphs requires $\Theta(nm)$ time with Brandes’s algorithm [Brandes 2001], where n is the number of nodes and m is the number of edges. Since this can be prohibitive for very large networks, several approximation algorithms exist in the literature [Borassi and Natale 2016; Geisberger et al. 2008; Riondato and Upfal 2014; Riondato and Kornaropoulos 2016]. Also for dynamic networks that evolve over time, such as social networks and the Web graph, recomputing betweenness at every time step can be too expensive. For this reason, a variety of dynamic algorithms have been proposed over the last years [Green et al. 2012; Kas et al. 2013; Lee et al. 2016; Nasre et al. 2014; Pontecorvi and Ramachandran 2015]. These algorithms usually keep track of the betweenness scores and additional information, such as the pairwise distances, and update them accordingly after a modification in the graph. Another problem that has recently been considered for betweenness and other centrality measures is the quick identification of the k most central nodes without computing the score of each node [Bergamini et al. 2016; Lee and Chung 2014].

There are several contexts in which having a high betweenness can be beneficial for the node itself. For example, in the field of transportation network analysis, the betweenness centrality seems to be positively related to the efficiency of an airport (see [Malighetti et al. 2009], where a network of 57 European airports has been analyzed). In a street network, increasing the betweenness of a shop or business would mean more traffic flowing through it and possibly more customers. In social networks, having high betweenness can be extremely beneficial for brokers [Everett and Valente 2016]. Therefore, an interesting problem is that of increasing the centrality of a given node. This problem has attracted considerable attention for page-rank [Crescenzi et al. 2015; Crescenzi et al. 2016], where much effort has been devoted to “fooling” search engines in order to increase the popularity of some web pages (an example is the well-known link farming [Wu and Davison 2005]). In addition to page-rank, the problem of increasing the centrality of a node has been considered also for other centrality measures, such as closeness centrality [Crescenzi et al. 2016] and eccentricity [Demaine and Zadimoghaddam 2010; Perumal et al. 2013].

In the above mentioned contexts, it is reasonable to assume that, in order to increase its betweenness, a node can only add edges incident to itself. Hence, in this paper we address the following problem: assuming that a node v can connect itself with k other nodes, how can we choose these nodes in order to maximize the betweenness centrality of v ? In other terms we want to add a set of k edges to the graph (all incident to v), such that the betweenness of v in the new graph is as high as possible. Since in some contexts one might be more interested in having a high ranking among other nodes rather than a high betweenness score, we also consider the case where we want to maximize the ranking increment of a node instead of its betweenness. We call such two optimization problems *maximum betweenness improvement* (MBI) and *maximum ranking improvement* (MRI), respectively.

Our contribution. We study both MBI and MRI problems in directed graphs. Our contribution can be summarized as follows: (i) We provide two hardness results, one for MBI and one for MRI. In particular, we prove that, unless $P = NP$, MBI cannot be approximated within a factor greater than $1 - \frac{1}{2e}$. Also, we show that, for any constant $\alpha \leq 1$, there is

no α -approximation algorithm for MRI, unless $P = NP$ (Section 4). (ii) We propose a greedy algorithm for MBI, which yields a $(1 - \frac{1}{\alpha})$ -approximation (Section 5). This is in contrast with the results for the undirected graph case, where it is known that the same algorithm has an unbounded approximation ratio [D’Angelo et al. 2016]. The complexity of the algorithm, if implemented naively, is $O(kn^2m)$. By means of experiments, we show that on directed random graphs, the approximation ratio (the ratio between the solution found by the optimum and the one found by our greedy algorithm) is never smaller than 0.96. Also, our experiments on real-world networks show that the greedy approach outperforms other heuristics, both in terms of betweenness improvement and ranking improvement. Although the approximation guarantee holds only for directed graphs, our tests show that the greedy algorithm works well also on undirected real-world networks. (iii) To make our greedy approach faster, we also develop a new algorithm for recomputing the betweenness centrality of a node after an edge insertion or a weight decrease (Section 6). The algorithm, which might be of independent interest, builds on a recent method for updating all-pairs shortest paths after an edge insertion [Slobbe et al. 2016]. In the worst case, our algorithm updates the betweenness of one node in $O(n^2)$ time, whereas all existing dynamic algorithms recompute the betweenness of all nodes in at least $O(nm)$ time. This is in contrast with the static case, where computing betweenness of all nodes is just as expensive as computing it for one node (at least, no algorithm exists that computes the betweenness of one node faster than for all nodes). In a context where the betweenness centrality of a single node needs to be recomputed, our experimental evaluation shows that our new algorithm is orders of magnitude faster than existing algorithms, on average by a factor 10^2 and up to a factor 10^3 . Also, using our dynamic algorithm, the worst-case complexity of our greedy approach for MBI decreases to $O(kn^3)$. However, our experiments show that it is actually much faster in practice. For example, we are able to target directed networks with hundreds of thousands of nodes in a few minutes.

2. RELATED WORK

Centrality improvement. In the following we describe the literature about algorithms that aim at optimizing some property of a graph by adding a limited number of edges. In [Meyerson and Tagiku 2009], the authors give a constant factor approximation algorithm for the problem of minimizing the average shortest-path distance between all pairs of nodes. Other works [Papagelis et al. 2011; Parotsidis et al. 2015] propose new algorithms for the same problem, experimentally show that they are good in practice. In [Bauer et al. 2012], the authors study the problem of minimizing the average number of hops in shortest paths of weighted graphs, prove that, unless $P = NP$, the problem cannot be approximated within a logarithmic factor, and propose two approximation algorithms with non-constant approximation guarantees. [Tong et al. 2012] and [Saha et al. 2015] focus on the problem of maximizing the leading eigenvalue of the adjacency matrix and give algorithms with proven approximation guarantees.

Some algorithms with proven approximation guarantees for the problem of minimizing the diameter of a graph are presented in [Bilò et al. 2012] and [Fрати et al. 2015].

In [Li and Yu 2015] and [Dehghani et al. 2015], the authors propose approximation algorithms with proven guarantees for the problem of making the number of triangles in a graph minimum and maximum, respectively. In [Papagelis 2015], the author studies the problem of minimizing the characteristic path length.

The problem analyzed in this paper differs from the above mentioned ones as it focuses on improving the centrality of a predefined vertex. Similar problems have been studied for other centrality measures, i.e. page-rank [Avrachenkov and Litvak 2006; Olsen and Viglas 2014], eccentricity [Demaine and Zadimoghaddam 2010; Perumal et al. 2013], average distance [Meyerson and Tagiku 2009], some measures related to the number of paths passing

through a given node [Ishakian et al. 2012], and closeness centrality [Crescenzi et al. 2015; Crescenzi et al. 2016].

The MBI problem has been studied for undirected weighted graphs and it has been proved that, in this case, the problem cannot be approximated within a factor greater than $1 - \frac{1}{2e}$, unless $P = NP$. Moreover, a natural greedy algorithm similar to the one presented in this paper for the directed case, exhibits an arbitrarily small approximation ratio [D’Angelo et al. 2016].

Dynamic algorithms for betweenness centrality. The general idea of dynamic betweenness algorithms is to keep track of the old betweenness values and to update them after some modification happens to the graph, which might be an edge or node insertion, an edge or node deletion, or a change in an edge’s weight. In particular, in case of edge insertions or weight decreases, the algorithms are often referred to as *incremental*, whereas for edge deletions or weight increases they are called *decremental*. All dynamic algorithms existing in the literature update the centralities of *all nodes* and most of them first update the distances and shortest paths between nodes and then recompute the fraction of shortest paths each node belongs to. The approach proposed by Green et al. [Green et al. 2012] for unweighted graphs maintains all previously calculated betweenness values and additional information, like the distance between each node pair and the list of *predecessors*, i.e. the nodes immediately preceding v in the shortest paths from s to v , for all node pairs (s, v) . Using this information, the algorithm limits the recomputation to the nodes whose betweenness has actually been affected. Kourtellis et al. [Kourtellis et al. 2015] modify the approach by Green et al. [Green et al. 2012] in order to reduce the memory requirements from $O(nm)$ to $O(n^2)$. Instead of storing the predecessors of each node v from each possible source, they recompute them every time the information is required.

Kas et al. [Kas et al. 2013] extend an existing algorithm for the dynamic all-pairs shortest paths (APSP) problem by Ramalingam and Reps [Ramalingam and Reps 1996] to also update BC scores. Nasre et al. [Nasre et al. 2014] compare the distances between each node pair before and after the update and then recompute the dependencies as in Brandes’s algorithm. Although this algorithm is faster than recomputation on some graph classes (i.e. when only edge insertions are allowed and the graph is sparse and weighted), it was shown in [Bergamini et al. 2015] that its performance in practice is always worse than that of the algorithm proposed in [Green et al. 2012]. Pontecorvi and Ramachandran [Pontecorvi and Ramachandran 2015] extend existing fully-dynamic APSP algorithms with new data structures to update *all* shortest paths and then recompute dependencies as in Brandes’s algorithm. Differently from the previous algorithms, the approach by Lee et al. [Lee et al. 2016] is not based on dynamic APSP algorithms, but splits the graph into biconnected components and then recomputes the betweenness values from scratch only within the component affected by the graph update. Although this allows for a smaller memory requirement ($\Theta(m)$ versus $\Omega(n^2)$ needed by the other approaches), the speedups on recomputation reported in [Lee et al. 2016] are significantly worse than those reported for example in [Green et al. 2012]. Recently, also dynamic algorithms that update an approximation of betweenness centrality have been proposed [Bergamini and Meyerhenke 2016; Hayashi et al. 2015; Riondato and Upfal 2016]. Notice that all existing dynamic algorithms update the betweenness of all nodes and their worst-case complexity is, in general, the same as static recomputation. This means, for exact algorithms, $O(nm)$ in unweighted and $O(n(m + n \log n))$ in weighted graphs.

Dynamic APSP algorithms. Since most of centrality metrics need shortest-paths computation, its efficient update is strictly related to the recomputation of shortest paths. As we mentioned before, several dynamic betweenness algorithms are based on dynamic APSP algorithms. The algorithm by Kas et al. [Kas et al. 2013] is based on the dynamic APSP algorithm by Ramalingam and Reps [Ramalingam and Reps 1996]. The worst-case running time

of this dynamic APSP algorithm is the same as recomputing from scratch, i.e. $O(nm)$ for unweighted and $O(n(m + n \log n))$ for weighted graphs. The first to propose an asymptotically faster APSP algorithm for fully-dynamic graphs (i.e. allowing all kinds of edge updates) with real edge weights were Demetrescu and Italiano [Demetrescu and Italiano 2004]. Their algorithm requires $O(n^2 \log^3 n)$ amortized time per update. This bound was then improved by Thorup [Thorup 2004], whose algorithm achieves $O(n^2(\log n + \log^2((m + n)/n)))$ amortized time per update. Nevertheless, the algorithm by Ramalingam and Reps [Ramalingam and Reps 1996] was shown to be the best-performing in practice in a subsequent experimental evaluation by Demetrescu and Italiano [Demetrescu and Italiano 2004]. Only recently, a new algorithm (QUINCA) for the incremental case only [Slobbe et al. 2016] has been proposed and shown to outperform the one by Ramalingam and Reps [Ramalingam and Reps 1996]. Because of its practical performance and its worst-case complexity of $O(n^2)$, we choose QUINCA as a building block for our new incremental algorithm for the betweenness of a single node. Section 6.1 contains a detailed description of QUINCA.

3. NOTATION AND PROBLEM STATEMENT

Let $G = (V, E)$ be a directed graph where $|V| = n$ and $|E| = m$. For each node v , N_v denotes the set of in-neighbors of v , i.e. $N_v = \{u \mid (u, v) \in E\}$. Given two nodes s and t , we denote by d_{st} , σ_{st} , and σ_{stv} the distance from s to t in G , the number of shortest paths from s to t in G , and the number of shortest paths from s to t in G that contain v , respectively. For each node v , the *betweenness centrality* [Freeman 1977] of v is defined as

$$b_v = \sum_{\substack{s, t \in V \\ s \neq t; s, t \neq v \\ \sigma_{st} \neq 0}} \frac{\sigma_{stv}}{\sigma_{st}}. \quad (1)$$

The *ranking* of each node v according to its betweenness centrality is defined as

$$r_v = |\{u \in V \mid b_u > b_v\}| + 1. \quad (2)$$

In this paper, we consider graphs that are augmented by adding a set S of arcs not in E . Given a set $S \subseteq V \times V \setminus E$ of arcs, we denote by $G(S)$ the graph augmented by adding the arcs in S to G , i.e. $G(S) = (V, E \cup S)$. For a parameter x of G , we denote by $x(S)$ the same parameter in graph $G(S)$, e.g. the distance from s to t in $G(S)$ is denoted as $d_{st}(S)$.

The betweenness centrality of a node might change if the graph is augmented with a set of arcs. In particular, adding arcs incident to some node v can increase the betweenness of v and its ranking. We are interested in finding a set S of arcs incident to a particular node v that maximizes $b_v(S)$. Therefore, we define the following optimization problem.

Maximum Betweenness Improvement (MBI)	
Given:	A directed graph $G = (V, E)$; a node $v \in V$; an integer $k \in \mathbb{N}$
Solution:	A set S of arcs incident to v , $S = \{(u, v) \mid u \in V \setminus N_v\}$, such that $ S \leq k$
Objective:	Maximize $b_v(S)$

Note that maximizing the betweenness value of a node v does not necessarily lead to maximizing the ranking position of v . Therefore, we also consider the problem of finding a set S of arcs incident to node v that maximizes *the increment of the ranking* of v with respect to its original ranking. We denote such an increment as $\rho_v(S)$, that is,

$$\rho_v(S) = r_v - r_v(S).$$

Informally, $\rho(S)$ represents the number of nodes that v “overtakes” by adding arcs in S to G . Therefore, we define the following optimization problem.

Maximum Ranking Improvement (MRI)	
Given:	A directed graph $G = (V, E)$; a vertex $v \in V$; and an integer $k \in \mathbb{N}$
Solution:	A set S of arcs incident to v , $S = \{(u, v) \mid u \in V \setminus N(v)\}$, such that $ S \leq k$
Objective:	Maximize $\rho_v(S)$

4. HARDNESS OF APPROXIMATION

In this section we first show that it is NP -hard to approximate problem MBI within a factor greater than $1 - \frac{1}{2e}$. Then, we focus on the MRI problem and show that it cannot be approximated within any constant bound, unless $P = NP$.

THEOREM 4.1. *Problem MBI cannot be approximated within a factor greater than $1 - \frac{1}{2e}$, unless $P = NP$.*

PROOF. We give an L-reduction with parameters a and b [Williamson and Shmoys 2011, Chapter 16] to the *maximum set coverage problem* (MSC) defined as follows: given a finite set X , a finite family \mathcal{F} of subsets of X , and an integer k' , find $\mathcal{F}' \subseteq \mathcal{F}$ such that $|\mathcal{F}'| \leq k'$ and $s(\mathcal{F}') = |\cup_{S_i \in \mathcal{F}'} S_i|$ is maximum. In detail, we will give a polynomial-time algorithm that transforms any instance I_{MSC} of MSC into an instance I_{MBI} of MBI and a polynomial-time algorithm that transforms any solution S_{MBI} for I_{MBI} into a solution S_{MSC} for I_{MSC} such that the following two conditions are satisfied for some values a and b :

$$OPT(I_{\text{MBI}}) \leq aOPT(I_{\text{MSC}}) \quad (3)$$

$$OPT(I_{\text{MSC}}) - s(S_{\text{MSC}}) \leq b(OPT(I_{\text{MBI}}) - b_v(S_{\text{MBI}})), \quad (4)$$

where OPT denotes the optimal value of an instance of an optimization problem. If the above conditions are satisfied and there exists an α -approximation algorithm A_{MBI} for MBI, then there exists a $(1 - ab(1 - \alpha))$ -approximation algorithm A_{MSC} for MSC [Williamson and Shmoys 2011, Chapter 16]. Since it is NP -hard to approximate MSC within a factor greater than $1 - \frac{1}{e}$ [Feige 1998], then the approximation factor of A_{MSC} must be smaller than $1 - \frac{1}{e}$, unless $P = NP$. This implies that $1 - ab(1 - \alpha) < 1 - \frac{1}{e}$ that is, the approximation factor α of A_{MBI} must satisfy $\alpha < 1 - \frac{1}{abe}$, unless $P = NP$. In the following, we give an L-reduction and determine the constant parameters a and b . In the reduction, each element x_i and each set S_j in an instance of MSC corresponds to a vertex in an instance of MBI, denoted by v_{x_i} and v_{S_j} , respectively. There is an arc from v_{x_i} to v_{S_j} if and only if $x_i \in S_j$. The MBI instance contains two further nodes v and t and an arc (v, t) . A solution to such an instance consists of arcs from nodes v_{S_j} to v and the aim is to cover with such arcs the maximum number of shortest paths from nodes v_{x_i} to t . We will prove that we can transform a solution to MBI into a solution to MSC such that any node v_{x_i} that has a shortest path passing through v corresponds to a covered element $x_i \in X$. We give more detail in what follows.

Given an instance $I_{\text{MSC}} = (X, \mathcal{F}, k')$ of MSC, where $\mathcal{F} = \{S_1, S_2, \dots, S_{|\mathcal{F}|}\}$, we define an instance $I_{\text{MBI}} = (G, v, k)$ of MBI, where:

- $G = (V, E)$;
- $V = \{v, t\} \cup \{v_{x_i} \mid x_i \in X\} \cup \{v_{S_j} \mid S_j \in \mathcal{F}\}$;
- $E = \{(v, t)\} \cup \{(v_{x_i}, v_{S_j}) \mid x_i \in S_j\}$;
- $k = k'$.

See Figure 1 for a visualization.

Without loss of generality, we can assume that any solution S_{MBI} to MBI contains only arcs (v_{S_j}, v) for some $S_j \in \mathcal{F}$. In fact, if a solution does not satisfy this property, then we can improve it in polynomial time by repeatedly applying the following transformation: for each arc $a = (v_{x_i}, v)$ in S_{MBI} such that $x_i \in X$, exchange a with an arc (v_{S_j}, v) such

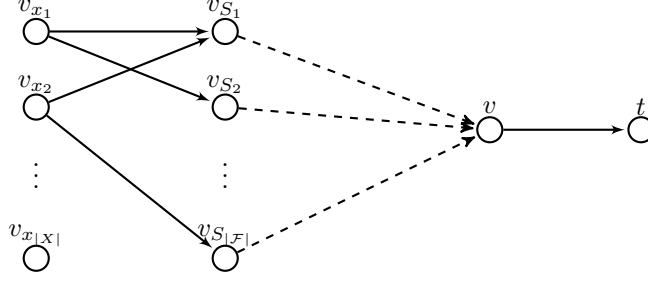


Fig. 1. Reduction used in Theorem 4.1. In the example $x_1 \in S_1$, $x_1 \in S_2$, $x_2 \in S_1$, and $x_2 \in S_{\mathcal{F}}$. The dashed arcs denote those added in a solution.

that $x_i \in S_j$ and $(v_{S_j}, v) \notin S_{\text{MBI}}$ if it exists or remove a otherwise. Note that if no arc (v_{S_j}, v) such that $x_i \in S_j$ and $(v_{S_j}, v) \notin S_{\text{MBI}}$ exists, then all the shortest paths from x_i to t pass through v and therefore the arc (v_{x_i}, v) can be removed without changing the value of $b_v(S_{\text{MBI}})$. Such a transformation does not decrease the value of $b_v(S_{\text{MBI}})$ in fact, all the shortest paths passing through v in the original solution still pass through v in the obtained solution. Moreover, if Condition (4) is satisfied for the obtained solution, then it is satisfied also for the original solution. In such a solution, all the paths (if any) from v_{x_i} to t , for each $x_i \in X$, and from v_{S_j} to t , for each $S_j \in \mathcal{F}$ pass through v and therefore the ratio $\frac{\sigma_{stv}(S_{\text{MBI}})}{\sigma_{st}(S_{\text{MBI}})}$ is 1, for each $s \in V \setminus \{v, t\}$ such that $\sigma_{st}(S_{\text{MBI}}) \neq 0$. We can further assume, again without loss of generality, that any solution S_{MBI} is such that $|S_{\text{MBI}}| = k$, in fact, if $|S_{\text{MBI}}| < k$, then we can add to S_{MBI} an arc (v_{S_j}, v) that is not yet in S_{MBI} . Note that such an arc must exist otherwise $k > |\mathcal{F}|$ and this operation does not decrease the value of $b_v(S_{\text{MBI}})$.

Given a solution $S_{\text{MBI}} = \{(v_{S_j}, v) \mid S_j \in \mathcal{F}\}$ to MBI, we construct the solution $S_{\text{MSC}} = \{S_j \mid (v_{S_j}, v) \in S_{\text{MBI}}\}$ to MSC. By construction, $|S_{\text{MSC}}| = |S_{\text{MBI}}| = k = k'$. Moreover, the set of elements x_i of X such that $\sigma_{v_{x_i}t}(S_{\text{MBI}}) \neq 0$ is equal to $\{x_i \in S_j \mid (v_{S_j}, v) \in S_{\text{MBI}}\} = \bigcup_{S_j \in S_{\text{MSC}}} S_j$. Therefore, the betweenness centrality of v in $G(S_{\text{MBI}})$ is:

$$\begin{aligned}
 b_v(S_{\text{MBI}}) &= \sum_{\substack{s \in V \setminus \{v, t\} \\ \sigma_{st}(S_{\text{MBI}}) \neq 0}} \frac{\sigma_{stv}(S_{\text{MBI}})}{\sigma_{st}(S_{\text{MBI}})} \\
 &= \sum_{\substack{x_i \in X \\ \sigma_{v_{x_i}t}(S_{\text{MBI}}) \neq 0}} \frac{\sigma_{v_{x_i}tv}(S_{\text{MBI}})}{\sigma_{v_{x_i}t}(S_{\text{MBI}})} + \sum_{\substack{S_j \in \mathcal{F} \\ \sigma_{v_{S_j}t}(S_{\text{MBI}}) \neq 0}} \frac{\sigma_{v_{S_j}tv}(S_{\text{MBI}})}{\sigma_{v_{S_j}t}(S_{\text{MBI}})} \\
 &= |\{x_i \in S_j \mid (v_{S_j}, v) \in S_{\text{MBI}}\}| + |\{S_j \mid (v_{S_j}, v) \in S_{\text{MBI}}\}| \\
 &= \left| \bigcup_{S_j \in S_{\text{MSC}}} S_j \right| + |S_{\text{MSC}}| \\
 &= s(S_{\text{MSC}}) + k.
 \end{aligned}$$

It follows that Conditions (3) and (4) are satisfied for $a = 2$, $b = 1$ since: $OPT(I_{\text{MBI}}) = OPT(I_{\text{MSC}}) + k \leq 2OPT(I_{\text{MSC}})$ and $OPT(I_{\text{MSC}}) - s(S_{\text{MSC}}) = OPT(I_{\text{MBI}}) - b_v(S_{\text{MBI}})$, where the first inequality is due to the fact that $OPT(I_{\text{MSC}}) \geq k$.¹ The statement follows by plugging the values of a and b into $\alpha < 1 - \frac{1}{abe}$. \square

¹If $OPT(I_{\text{MSC}}) < k$, then the greedy algorithm finds an optimal solution for MSC.

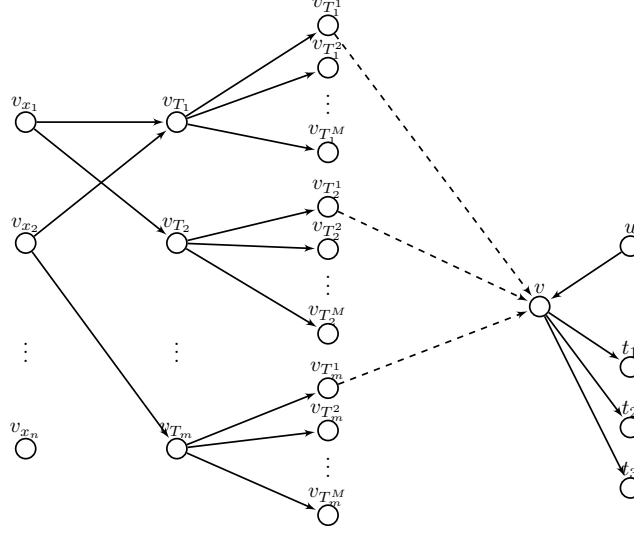


Fig. 2. The reduction used in Theorem 4.2. The dashed arcs denote those added in a solution to MRI.

In the next theorem, we show that, unless $P = NP$, we cannot find a polynomial time approximation algorithm for MRI with a constant approximation guarantee.

THEOREM 4.2. *For any constant $\alpha \leq 1$, there is no α -approximation algorithm for the MRI problem, unless $P = NP$.*

PROOF. By contradiction, let us assume that there exists a polynomial time algorithm A that guarantees an approximation factor of α . We show that we can use A to determine whether an instance I of the *exact cover by 3-sets* problem (X3C) admits a feasible solution or not. Problem X3C is known to be NP -complete [Garey and Johnson 1979] and therefore this implies a contradiction. In the X3C problem we are given a finite set X with $|X| = 3q$ and a collection C of 3-element subsets of X and we ask whether C contains an exact cover for X , that is, a subcollection $C' \subseteq C$ such that every element of X occurs in exactly one member of C' . Note that we can assume without loss of generality that $m > q$.

Given an instance $I = (X, C)$ of X3C where $|X| = n = 3q$ and $|C| = m$, we define an instance $I' = (G, v, k)$ of MRI as follows.

- $G = (V, E)$;
- $V = \{v, u, t_1, t_2, t_3\} \cup \{v_{x_i} \mid x_i \in X\} \cup \{v_{T_j} \mid T_j \in C\} \cup \{v_{T_j^\ell} \mid T_j \in C, \ell = 1, 2, \dots, M\}$;
- $E = \{(v_{x_i}, v_{T_j}) \mid x_i \in T_j\} \cup \{(v_{T_j}, v_{T_j^\ell}) \mid T_j \in C, \ell = 1, 2, \dots, M\} \cup \{(u, v), (v, t_1), (v, t_2), (v, t_3)\}$;
- $k = q$.

where $M = 5q + 1$. See Figure 2 for a visualization.

The proof proceeds by showing that I admits an exact cover if and only if I' admits a solution S such that $\rho_v(S) > 0$. This implies that, if OPT is an optimal solution for I' , then $\rho_v(OPT) > 0$ if and only if I admits an exact cover. Hence, the statement follows by observing that algorithm A outputs a solution S such that $\rho_v(S) > \alpha \rho_v(OPT)$ and hence $\rho_v(S) > 0$ if and only if I admits an exact cover.

In I' , $b_v = 3$, $b_{v_{T_j}} = 3M = 15q + 3$, for each $T_j \in C$, and $b_w = 0$, for any other node w . Therefore, $r_{T_j} = 1$, for each $T_j \in C$, $r_v = m + 1$, and $r_w = m + 2$, for any other node w . In the proof we will use the observation that, in instance I' , adding arcs incident to v does

not decrease the betweenness value of any node, that is for any node $w \in V$ and for any solution S to I' , $b_w(S) \geq b_w$.

If instance I of X3C admits an exact cover C' , then consider the solution $S = \{(v_{T_j^1}, v) \mid T_j \in C'\}$ to I' . Note that $|S| = q = k$ and therefore we only need to show that $\rho_v(S) > 0$. Indeed, in the following we show that $\rho_v(S) = m - q > 0$. Since C' is an exact cover, then all nodes v_{x_i} are connected to the 3 nodes t_i and all the paths connecting them pass through v . The same holds for nodes v_{T_j} and $v_{T_j^1}$ such that $T_j \in C'$. Since there are $3q$ nodes v_{x_i} , q nodes v_{T_j} such that $T_j \in C'$, and q nodes $v_{T_j^1}$ such that $T_j \in C'$, then the betweenness centrality of v increases to $b_v(S) = 3(5q+1) = 15q+3$. Nodes v_{T_j} and $v_{T_j^1}$ such that $T_j \in C'$ increase their centrality to $b_{v_{T_j}}(S) = 3(M+4) = 15q+15$ and $b_{v_{T_j^1}}(S) = 16$, respectively. Any other node does not change its betweenness centrality. Therefore the only nodes that have a betweenness higher than v are the q nodes $v_{T_j^1}$ such that $T_j \in C'$. It follows that $r_v(S) = q+1$ and $\rho_v(S) = m+1 - (q+1) = m-q > 0$.

Let us now assume that I' admits solution S such that $|S| \leq k$ and $\rho_v(S) > 0$. We first prove that S is only made of arcs in the form $(v_{T_j^1}, v)$ and that $b_v(S) \geq 15q+3$ or that it can be transformed in polynomial time into a solution with such a form without increasing its size. Assume that S has arcs not in this form, then we can apply one of the following transformations to each of such arcs $e = (w, v)$.

- If $w = v_{x_i}$ for some $x_i \in X$ and there exists a node $v_{T_j^1}$ such that $x_i \in T_j$ and $(v_{T_j^1}, v) \notin S$, then remove e and add arc $(v_{T_j^1}, v)$ to S ;
- If $w = v_{x_i}$ for some $x_i \in X$ and $(v_{T_j^1}, v) \in S$ for all T_j such that $x_i \in T_j$, then remove e ;
- If $w = v_{T_j}$ for some $T_j \in C$ and $(v_{T_j^1}, v) \notin S$, then remove e and add arc $(v_{T_j^1}, v)$ to S ;
- If $w = v_{T_j}$ for some $T_j \in C$ and $(v_{T_j^1}, v) \in S$, then remove e ;
- If $w = v_{T_j^i}$ for some $T_j \in C$ and $i > 1$, and $(v_{T_j^1}, v) \notin S$, then remove e and add arc $(v_{T_j^1}, v)$ to S ;
- If $w = v_{T_j^i}$ for some $T_j \in C$ and $i > 1$, and $(v_{T_j^1}, v) \in S$, then remove e and add arc $(v_{T_j^1}, v)$ to S for some j' such that $(v_{T_j^1}, v) \notin S$;²
- If $w = t_i$ for $i \in \{1, 2, 3\}$, then remove e and add arc $(v_{T_j^1}, v)$ to S for some j' such that $(v_{T_j^1}, v) \notin S$.²

Let us denote by S' and S the original solution and the solution that is eventually obtained by applying the above transformations, respectively. All the above transformations remove an arc and possibly add another arc, therefore the size of the transformed solution is at most the original size, that is $|S| \leq |S'| \leq k$. It remains to show that $\rho_v(S') > 0$ implies $b_v(S) \geq 15q+3$. Indeed, observe that v is initially in position $m+1$ and the only nodes that have a betweenness value higher than v are the m nodes v_{T_j} . Therefore, since $\rho_v(S') > 0$, there is at least a node v_{T_j} such that $b_v(S') \geq b_{v_{T_j}}(S')$. Moreover, all the transformations do not decrease the value of b_v and then $b_v(S) \geq b_v(S')$ and, considering that $b_{v_{T_j}}(S') \geq b_{v_{T_j}} = 15q+3$, we obtain $b_v(S) \geq 15q+3$.

We now prove that the solution $C' = \{T_j \mid (v_{T_j^1}, v) \in S\}$ to I is an exact cover. By contradiction, let us assume that an element in X is not contained in any set in C' or that an element in X is contained in more than one set in C' . The latter case implies the former one since $|C'| = q$, all the sets in C' contain exactly 3 elements, and $|X| = 3q$. Hence, we

²Note that such j' must exists, otherwise $m < q$.

ALGORITHM 1: GREEDY algorithm.**Input** : A directed graph $G = (V, E)$; a vertex $v \in V$; and an integer $k \in \mathbb{N}$ **Output:** Set of edges $S \subseteq \{(u, v) \mid u \in V \setminus N_v\}$ such that $|S| \leq k$

```

1  $S \leftarrow \emptyset$ ;
2 for  $i = 1, 2, \dots, k$  do
3   foreach  $u \in V \setminus (N_v(S))$  do
4      $\mid$  Compute  $b_v(S \cup \{(u, v)\})$ 
5    $u_{\max} \leftarrow \arg \max \{b_v(S \cup \{(u, v)\}) \mid u \in V \setminus (N_v(S))\}$ ;
6    $S \leftarrow S \cup \{(u_{\max}, v)\}$ ;
7 return  $S$ ;
```

assume that an element in $|X|$ is not contained in any set in C' . This implies that there exists a node $v_{x_i} \in V$ that has no path to nodes t_i and therefore the betweenness of v is at most $3(1 + 3q - 1 + 2q) = 15q$, which is a contradiction to $b_v(S) \geq 15q + 3$. \square

5. GREEDY APPROXIMATION ALGORITHM FOR MBI

In this section we propose an algorithm that guarantees a constant approximation ratio for the MBI problem. The algorithm exploits the results of Nemhauser et al. on the approximation of monotone submodular objective functions [Nemhauser et al. 1978]. Let us consider the following optimization problem: given a finite set N , an integer k' , and a real-valued function z defined on the set of subsets of N , find a set $S \subseteq N$ such that $|S| \leq k'$ and $z(S)$ is maximum. If z is *monotone and submodular*³, then the following greedy algorithm exhibits an approximation of $1 - \frac{1}{e}$ [Nemhauser et al. 1978]: start with the empty set and repeatedly add an element that gives the maximal marginal gain, that is, if S is a partial solution, choose the element $j \in N \setminus S$ that maximizes $z(S \cup \{j\})$.

THEOREM 5.1 ([NEMHAUSER ET AL. 1978]). *For a non-negative, monotone submodular function z , let S be a set of size k obtained by selecting elements one at a time, each time choosing an element that provides the largest marginal increase in the value of z . Then S provides a $(1 - \frac{1}{e})$ -approximation.*

In this paper we exploit such results by showing that b_v is monotone and submodular with respect to the possible set of arcs incident to v . Hence, we define a greedy algorithm, reported in Algorithm 1, that provides a $(1 - \frac{1}{e})$ -approximation. Algorithm 1 iterates k times and, at each iteration, it adds to a solution S an arc (u, v) that, when added to $G(S)$, gives the largest marginal increase in the betweenness of v , that is, $b_v(S \cup \{(u, v)\})$ is maximum among all the possible arcs not in $E \cup S$ incident to v . The next theorem shows that the objective function is monotone and submodular.

THEOREM 5.2. *For each node v , function b_v is monotone and submodular with respect to any feasible solution for MBI.*

PROOF. We prove that each term of the sum in the formula of b_v is monotone increasing and submodular. For each pair $s, t \in V$ such that $s \neq t$ and $s, t \neq v$, we denote such term by $b_{stv}(X) = \frac{\sigma_{stv}(X)}{\sigma_{st}(X)}$, for each solution X to MBI.

We first give two observations that will be used in the proof. Let X, Y be two solutions to MBI such that $X \subseteq Y$.

— Any shortest path from s to t in $G(X)$ exists also in $G(Y)$. It follows that $d_{st}(Y) \leq d_{st}(X)$.

³For a ground set N , a function $z : 2^N \rightarrow \mathbb{R}$ is submodular if for any pair of sets $S \subseteq T \subseteq N$ and for any element $e \in N \setminus T$, $z(S \cup \{e\}) - z(S) \geq z(T \cup \{e\}) - z(T)$.

- If $d_{st}(Y) < d_{st}(X)$, then any shortest path from s to t in $G(Y)$ pass through arcs in $Y \setminus X$. Therefore, all such paths pass through v . It follows that if $d_{st}(Y) < d_{st}(X)$, then $b_{stv}(Y) = 1$.

We now show that b_v is monotone increasing, that is for each solution S to MBI and for each node u such that $(u, v) \notin S \cup E$,

$$b_{stv}(S \cup \{(u, v)\}) \geq b_{stv}(S).$$

If $d_{st}(S) > d_{st}(S \cup \{(u, v)\})$, then $b_{stv}(S \cup \{(u, v)\}) = 1$ and since by definition $b_{stv}(S) \leq 1$, then the statement holds. If $d_{st}(S) = d_{st}(S \cup \{(u, v)\})$, then either (u, v) does not belong to any shortest path from s to t and then $b_{stv}(S \cup \{(u, v)\}) = b_{stv}(S)$, or (u, v) belongs to a newly added shortest path from s to t with the same weight and $b_{stv}(S \cup \{(u, v)\}) = \frac{\sigma_{stv}(S) + \delta}{\sigma_{st}(S) + \delta} > \frac{\sigma_{stv}(S)}{\sigma_{st}(S)} = b_{stv}(S)$, where $\delta \geq 1$ is the number of shortest paths from s to t that pass through arc (u, v) in $G(S \cup \{(u, v)\})$. In any case the statement holds.

We now show that b_{stv} is submodular, that is for each pair of solutions to MBI S, T such that $S \subseteq T$ and for each node u such that $(u, v) \notin T \cup E$,

$$b_{stv}(S \cup \{(u, v)\}) - b_{stv}(S) \geq b_{stv}(T \cup \{(u, v)\}) - b_{stv}(T).$$

We analyze the following cases:

- $d_{st}(S) > d_{st}(T)$. In this case, $b_{stv}(T \cup \{(u, v)\}) - b_{stv}(T) = 0$ since in any case $b_{stv}(T \cup \{(u, v)\}) = b_{stv}(T) = 1$. As b_{stv} is monotone increasing, then $b_{stv}(S \cup \{(u, v)\}) - b_{stv}(S) \geq 0$.
- $d_{st}(S) = d_{st}(T)$.
 - $d_{st}(S) > d_{st}(S \cup \{(u, v)\})$. In this case, $d_{st}(T) > d_{st}(T \cup \{(u, v)\})$ and $b_{stv}(T \cup \{(u, v)\}) = b_{stv}(S \cup \{(u, v)\}) = 1$. Moreover $b_{stv}(T) \geq b_{stv}(S)$. Therefore $b_{stv}(S \cup \{(u, v)\}) - b_{stv}(S) \geq b_{stv}(T \cup \{(u, v)\}) - b_{stv}(T)$.
 - $d_{st}(S) = d_{st}(S \cup \{(u, v)\})$. In this case $d_{st}(T) = d_{st}(T \cup \{(u, v)\})$. Let us denote $b_{stv}(S) = \frac{\alpha}{\beta}$, then we have that $b_{stv}(T) = \frac{\alpha + \gamma}{\beta + \gamma}$, $b_{stv}(S \cup \{(u, v)\}) = \frac{\alpha + \delta}{\beta + \delta}$, and $b_{stv}(T \cup \{(u, v)\}) = \frac{\alpha + \gamma + \delta}{\beta + \gamma + \delta}$, where γ and δ are the number of shortest paths between s and t in $G(T)$ that pass through arcs in $T \setminus S$ and arc (u, v) , respectively. The statement follows since $\frac{\alpha + \delta}{\beta + \delta} - \frac{\alpha}{\beta} \geq \frac{\alpha + \gamma + \delta}{\beta + \gamma + \delta} - \frac{\alpha + \gamma}{\beta + \gamma}$ for any $\alpha \leq \beta$, i.e. $\sigma_{stv}(S) \leq \sigma_{st}(S)$. \square

COROLLARY 5.3. *Algorithm 1 provides a $(1 - \frac{1}{e})$ -approximation for the MBI problem.*

It is easy to compute the computational complexity of Algorithm GREEDY. Line 2 iterates over all the numbers from 1 to k . Then, in Line 3, all the nodes u that are not yet neighbors of v are scanned. The number of these nodes is clearly $O(n)$. Finally, in Line 4, for each node u in Line 3, we add the edge $\{u, v\}$ to the graph and compute the betweenness in the new graph. Since computing betweenness requires $O(nm)$ operations in unweighted graphs, the total running time of GREEDY is $O(kn^2m)$. In Section 6 we show how to decrease this running time to $O(kn^3)$ by using a dynamic algorithm for the computation of betweenness centrality at Line 4.

5.1. Experimental evaluation

In this section we evaluate GREEDY in terms of accuracy and we compare it both with the optimum and with some alternative baselines. All algorithms compared in our experiments are implemented in C++, building on the open-source NetworkKit [Staudt et al. pear] framework. The experiments were done on a machine equipped with 256 GB RAM and a 2.7 GHz Intel Xeon CPU E5-2680 having 2 sockets with 8 cores each. To make the comparison with previous work more meaningful, we use only one of the 16 cores. The machine runs 64 bit SUSE Linux and we compiled our code with g++-4.8.1 and OpenMP 3.1.

To speed up the computation of GREEDY (and therefore to target larger graphs), we do not recompute betweenness from scratch in Line 4 of Algorithm 1, but we use the dynamic algorithm that will be described in Section 6. Notice that this does not affect the solution found by the algorithm, only its running time, which will be reported in Section 6.4. Since computing the optimum by examining all possible k -tuples would be too expensive even on small graphs, we use an Integer Programming (IP) formulation, described in the following paragraph.

5.1.1. IP formulation for MBI on directed graphs. Let S be a solution to an instance of MBI. Given a node v , we define a variable x_u for each node $u \in V \setminus (N_v \cup \{v\})$

$$x_u = \begin{cases} 1 & \text{if } (u, v) \in S \\ 0 & \text{otherwise.} \end{cases}$$

We define a variable y_{st} for each $s, t \in V \setminus \{v\}$, $s \neq t$.

$$y_{st} = \begin{cases} 1 & \text{If all shortest paths from } s \text{ to } t \text{ in } G(S) \text{ pass through node } v \\ 0 & \text{otherwise.} \end{cases}$$

For each pair of nodes $s, t \in V \setminus \{v\}$, $s \neq t$, we denote by $A(s, t)$ the set of nodes u not in N_v such that all the shortest paths between s and t in $G(\{(u, v)\})$ pass through edge (u, v) and hence through node v . Note that in this case, $d_{st} > d_{st}(\{(u, v)\})$ and hence $A(s, t)$ is defined as $A(s, t) = \{u \mid d_{st} > d_{st}(\{(u, v)\})\}$. Set $B(s, t)$ is defined as the set of nodes u not in N_v such that at least a shortest path between s and t in $G(\{(u, v)\})$ does not pass through edge (u, v) and hence $B(s, t) = V \setminus (A(s, t) \cup N_v)$. We denote by $\bar{\sigma}_{stv}(u)$ the number of shortest paths from s to t in $G(\{(u, v)\})$ passing thorough edge (u, v) .

The following non linear formulation solves the MBI problem:

$$\max \sum_{\substack{s, t \in V \\ s \neq t; s, t \neq v}} \left((1 - y_{st}) \frac{\sigma_{stv} + \sum_{u \in B(s, t)} x_u \bar{\sigma}_{stv}(u)}{\sigma_{st} + \sum_{u \in B(s, t)} x_u \bar{\sigma}_{stv}(u)} + y_{st} \right) \quad (5)$$

$$\text{subject to } \sum_{u \in A(s, t)} x_u \geq y_{st}, \quad s, t \in V \setminus \{v\}, s \neq t \quad (6)$$

$$\begin{aligned} \sum_{u \in V \setminus (N_v \cup \{v\})} x_u &\leq k, \\ x_u, y_{st} &\in \{0, 1\} \end{aligned} \quad s \in V \setminus \{v\}, t \in V \setminus \{v, s\}$$

Let us consider a solution S to the above formulation. In the case that $y_{st} = 1$, for some pair of nodes $s, t \in V \setminus \{v\}$, $s \neq t$, then Constraint (6) implies that, for at least a node $u \in A(s, t)$, variable x_u must be set to 1 and hence all the shortest paths between s and t in $G(S)$ pass through v . In this case, the term corresponding to pair (s, t) in the objective function (5) is correctly set to be equal to 1.

If $y_{st} = 0$ and $x_u = 0$, for each $u \in A(s, t)$, then the number of shortest paths between s and t in $G(S)$ passing trough v is equal to $\sigma_{stv} + \sum_{u \in B(s, t)} x_u \bar{\sigma}_{stv}(u)$ and the overall number of shortest paths between s and t in $G(S)$ is equal to $\sigma_{st} + \sum_{u \in B(s, t)} x_u \bar{\sigma}_{stv}(u)$. In this case, the term corresponding to pair (s, t) in the objective function (5) is correctly set to be equal to $\frac{\sigma_{stv} + \sum_{u \in B(s, t)} x_u \bar{\sigma}_{stv}(u)}{\sigma_{st} + \sum_{u \in B(s, t)} x_u \bar{\sigma}_{stv}(u)}$.

Note that, $\frac{\sigma_{stv} + \sum_{u \in B(s, t)} x_u \bar{\sigma}_{stv}(u)}{\sigma_{st} + \sum_{u \in B(s, t)} x_u \bar{\sigma}_{stv}(u)} \leq 1$ and therefore a solution in which $y_{st} = 0$ and $x_u = 1$, for some $u \in A(s, t)$ is at least as good as a solution in which y_{st} is set to 1 instead

of 0 and the other variables are unchanged. Hence, we can assume without loss of generality that the case in which $y_{st} = 0$ and $x_u = 1$, for some $u \in A(s, t)$, cannot occur in an optimal solution.

We solve the program with the Mixed-Integer Nonlinear Programming Solver Couenne [Belotti et al. 2009] and measure the approximation ratio of the greedy algorithm on three types of randomly generated directed networks, namely directed Preferential Attachment (in short, PA) [Bollobás et al. 2003], Copying (in short, COPY) [Kumar et al. 2000], Compressible Web (in short, COMP) [Chierichetti et al. 2009]. For each graph type, we generate 5 different instances with the same size. We focus our attention on twenty vertices v , which have been chosen on the basis of their original betweenness ranking. In particular, we divide the list of vertices, sorted by their original ranking, in four intervals, and choose five random vertices uniformly at random in each interval. In each experiment, we add $k = \{1, 2, \dots, 7\}$ edges. We evaluate the quality of the solution produced by the greedy algorithm by measuring its approximation ratio and we report the results in Table I. The

Table I. Comparison between the GREEDY algorithm and the optimum. The first three columns report the type and size of the graphs; the fourth column reports the approximation ratio.

Graph	Nodes	Edges	Min. approx. ratio
PA	100	130	1
COPY	100	200	0.98
COMP	100	200	0.98
COMP	100	500	0.96

experiments clearly show that the experimental approximation ratio is by far better than the theoretical one proven in the previous section. In fact, in all our tests, the experimental ratio is always greater than 0.96.

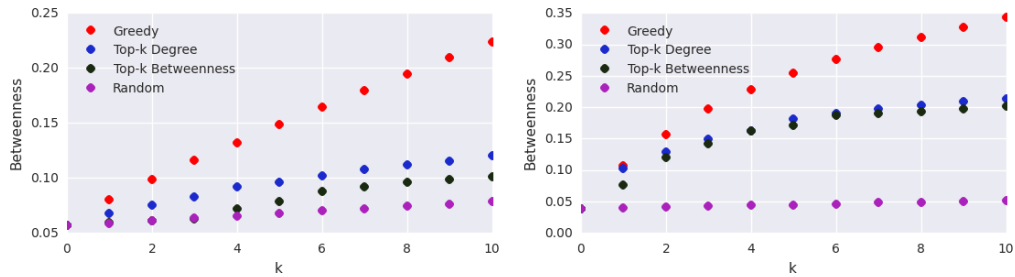


Fig. 3. Percentage betweenness of the pivot as a function of the number k of inserted edges for the four heuristics. Left: results for the `munmun-digg-reply` graph. Right: results for the `linux` graph.

5.1.2. Results for real-world directed networks. We also analyze the performance of GREEDY on the real-world directed networks of Table II (Section 6.4). Since finding the optimum on these networks would take too long, we compare the solution of GREEDY with the following three baselines:

- TOP-K DEGREE: the algorithm that connects the k nodes having the highest degree to v ;
- TOP-K BETWEENNESS: the algorithm that connects the k nodes having the highest betweenness centrality to v ;
- RANDOM: the algorithm that connects k nodes extracted uniformly at random to v .

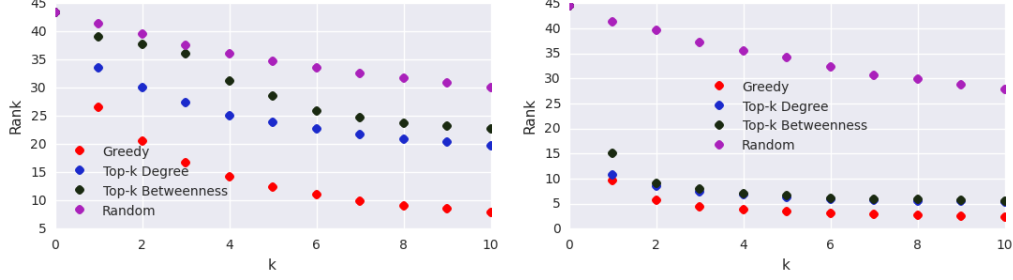


Fig. 4. Percentage rank of the pivot as a function of the number k of inserted edges for the four heuristics. Left: results for the `munmun-digg-reply` graph. Right: results for the `linux` graph.

For each graph, we pick one node at random, compute its betweenness on the initial graph and try to increase it with the four heuristics. We refer to the selected node as *pivot*. Since the results may vary depending on the initial betweenness of the pivot, we also repeat each experiment with 10 different pivots and report the average results over the different pivots. In each experiment, we add $k = \{1, 2, \dots, 10\}$ edges and compute the ranking and betweenness of the pivot after each insertion. Figure 3 reports the results for two directed graphs, namely `munmun-digg-reply` (left) and `linux` (right). We define the percentage betweenness of a node v as $b_v \cdot \frac{100}{(n-1)(n-2)}$, where b_v is the betweenness of v and $(n-1)(n-2)$ represents the maximum theoretical betweenness a node can have in a graph with n nodes. For each value of k , the plots show the average percentage betweenness of a pivot after the insertion of k edges (each point represents the average over the 10 pivots). Clearly, the pivot's betweenness after k insertions is a non-decreasing function of k , since the insertion of an edge can only increase (or leave unchanged) the betweenness of one of its endpoints. In both plots, GREEDY outperforms the other heuristics. For example, after k edge insertions, the average betweenness of a pivot in the `munmun-digg-reply` graph is 81460 with GREEDY, 43638 with TOP-K DEGREE, 36690 with TOP-K BETWEENNESS and 28513 with RANDOM. A similar behavior can be observed for the average ranks of the pivots, reported in Figure 4. The figures report the percentage ranks, i.e. the ranks multiplied by $\frac{100}{n}$, since n is the maximum rank a node can have in a graph with n nodes. This can be seen as the fraction of nodes with higher betweenness than the pivot. On `munmun-digg-reply`, the average initial rank is 2620 (about 43%). After 10 insertions, the rank obtained using GREEDY is 476 (about 7%), whereas the one obtained by the other approaches is never lower than 1188 (about 19%). It is interesting to notice that 3 edge insertions with GREEDY yield a rank of 1011, which is better than the one obtained by the other approaches after 10 insertions. Similarly, also on the `linux` graph, 3 iterations of GREEDY are enough to bring down the rank from 2498 (45.6%) to 247 (4.4%), whereas the other approaches cannot go below 299 (5.3%) with 10 iterations. Similar results can be observed on the other tested (directed) instances. Figure 5 reports the average results over all directed networks, both in terms of betweenness (left) and rank (right) improvement. The initial average betweenness of the sample pivots is 0.015%. GREEDY is by far the best approach, with an average final percentage betweenness (after 10 iterations) of 0.38% and an average final percentage rank of 1.4%. As a comparison, the best alternative approach (TOP-K DEGREE) yields a percentage betweenness of 0.22% and a percentage rank of 7.3%. Not surprisingly, the worst approach is RANDOM, which in 10 iterations yields a final percentage betweenness of 0.04% and an average percentage rank of 10.2%. On average, a single iteration of GREEDY is sufficient for a percentage rank of 5.5%, better than the one obtained by all other approaches in 10 iterations. Also, it is interesting to notice that in our experiments TOP-K DEGREE performs

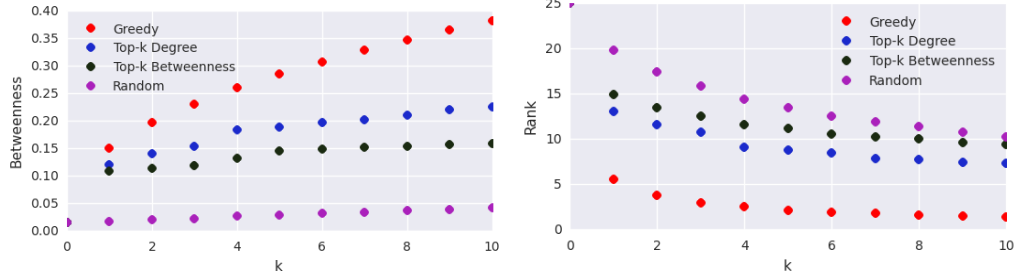


Fig. 5. Average results over all directed networks. On the left, average percentage betweenness of the pivots as a function of k . On the right, average percentage rank of the pivots.

significantly better than TOP-K BETWEENNESS. This means that, for the betweenness of a node in a directed graph, it is more important to have incoming edges from nodes with high out-degree than with high betweenness. We will see in the following that our results show a different behavior for undirected graphs.

Also, notice that, although the percentage betweenness scores are quite low, the improvement using GREEDY is still large: with 10 insertions, on average the scores change from an initial 0.015% to 0.38%, which is about 25 times the initial value.

5.1.3. Results for real-world undirected graphs. Although it was proven [D’Angelo et al. 2016] that GREEDY has an unbounded approximation ratio for undirected graphs, it is still not clear how it actually performs in practice. Therefore, we compare GREEDY with TOP-K BETWEENNESS, TOP-K DEGREE and RANDOM also on several undirected real-world networks, listed in Table III of Section 6.4. Figure 6 shows the percentage betweenness and

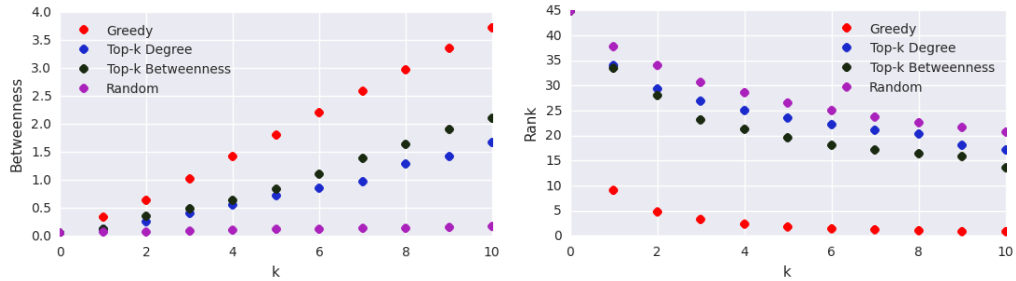


Fig. 6. Average results over all undirected networks. On the left, average percentage betweenness of the pivots as a function of k . On the right, average percentage rank of the pivots.

ranking, averaged over the undirected networks of Table III. Also in this case, GREEDY outperforms the other heuristics. In particular, the average initial betweenness of the pivots in the different graphs is 0.05%. After 10 iterations, the betweenness goes up to 3.7% with GREEDY, 1.6% with TOP-K DEGREE, 2.1% with TOP-K BETWEENNESS and only 0.17% with RANDOM. The average initial rank is 45%. GREEDY brings it down to 0.7% with ten iterations and below 5% already with two. Using the other approaches, the average rank is always worse than 10% for TOP-K BETWEENNESS, 15% for TOP-K DEGREE and 20% for RANDOM. As mentioned before, differently from directed graphs, TOP-K BETWEENNESS performs significantly better than TOP-K DEGREE in undirected graphs.

Also, notice that in undirected graphs the percentage betweenness scores of the nodes in the examined graphs are significantly larger than those in the directed graphs. This could be due to the fact that many node pairs have an infinite distance in the examined directed graphs, meaning that these pairs do not contribute to the betweenness of any node. Also, say we want to increase the betweenness of x by adding edge (v, x) . The pairs (s, t) for which we can have a shortcut (leading to an increase in the betweenness of x) are limited to the ones such that s can reach v and such that t is reachable from x , which might be a small fraction of the total number of pairs. On the contrary, most undirected graphs have a giant connected component containing the greatest majority of the nodes. Therefore, it is very likely that a pivot belongs to the giant component or that it will after the first edge insertion.

It is interesting to notice that, despite the unbounded approximation ratio, the improvement achieved by GREEDY on undirected graphs is even larger than for the directed ones: on average 74 times the initial score.

6. DYNAMIC ALGORITHM FOR BETWEENNESS CENTRALITY OF A SINGLE NODE

Algorithm 1 requires to add edges to the graph and to recompute the betweenness centrality b_v of node v after each edge insertion. Instead of recomputing it from scratch every time, we use a dynamic algorithm. The idea is to keep track of information regarding the graph and just update the parts that have changed as a consequence of the edge insertion. As described in Section 2, several algorithms for updating betweenness centrality after an edge insertion have been proposed. However, these algorithms update the betweenness of *all nodes*, whereas in Algorithm 1 we are interested in the betweenness of a *single node*. In this case, using an algorithm that recomputes the betweenness of all nodes would require a significant amount of superfluous operations. Let us consider the example shown in Figure 7.

The insertion of an edge (u, v) does not only affect the betweenness of the nodes lying in the new shortest paths, but also that of the nodes lying in the old shortest paths between affected sources and affected targets (represented in red). Indeed, the fraction of shortest paths going through these nodes (and therefore their betweenness) has decreased as a consequence of the new insertion. Therefore, algorithms for updating the betweenness of all nodes have to walk over each old shortest path between node pairs whose distance has changed. However, we will show that if we are only interested in the betweenness of one particular node x , we can simply update the distances (and number of shortest paths) and check which of these updates affect the betweenness of x . Section 6.2 describes our new dynamic algorithm for updating the betweenness of a single node after an edge insertion. Notice that the algorithm could be used in any context where one needs to keep track of the betweenness of a single node after an edge insertion (or weight decrease) and not only for the betweenness improvement. Since our new algorithm builds on a recent algorithm for APSP update called QUINCA, we first describe QUINCA in Section 6.1 and then explain how this can be extended and adapted to recompute betweenness in Section 6.2.

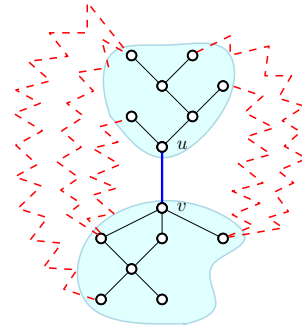


Fig. 7. Insertion of edge (u, v) affects the betweenness of nodes lying in the old shortest paths (red).

6.1. QUINCA algorithm for incremental APSP

QUINCA [Slobbe et al. 2016] recomputes the APSP after an edge insertion or an edge weight decrease. For simplicity, in the following we only consider the case of edge insertion in unweighted graphs and refer the reader to [Slobbe et al. 2016] for extensions to weighted graphs and weight decreases.

Let us assume a new edge (u, v) is inserted into the graph. An edge insertion can either create shortcuts in the shortest paths or leave them unchanged, which means that the new distance d'_{st} between any two nodes s and t can only be smaller than or equal to the old distance d_{st} .

Let us name *affected pairs* the node pairs (s, t) such that $d_{st} > d'_{st} = d_{su} + 1 + d_{vt}$. Also, let the *affected sources* of v be the set $S(v)$ of nodes $\{s \in V : d_{sv} > d'_{sv} = d_{su} + 1\}$ and let the set $T(u)$ of *affected targets* of u be $T(u) = \{t \in V : d_{ut} > d'_{ut} = 1 + d_{vt}\}$. In [Slobbe et al. 2016] it was proven that if (s, t) is an affected node pair, then $s \in S(v)$ and $t \in T(u)$. This allows us to reduce the search space of the affected pairs to the nodes whose distance to v or from u has changed. Also, let a *predecessor* in a shortest path from v to t be any node x such that $d_{vt} = d_{vx} + d_{xt}$. It was shown [Slobbe et al. 2016] that if (s, t) is an affected pair, then also (s, x) is an affected pair, for each x predecessor of t in a shortest path from v to t . Figure 8 explains this concept. The insertion of edge (u, v) creates a shortcut between s and t , making (s, t) an affected pair. Similarly, the new edge creates a shortcut between s and each predecessor of t in the shortest path from v , i.e. v and x . This property is the basis for Algorithm 18, describing QUINCA. First, we identify the set of affected sources of v (Line 1). This can be easily done by starting a BFS rooted in u and interrupting it when all non-visited nodes w are not affected (i.e. $d_{wv} \leq d_{wu} + 1$). After this, we can find the affected targets by running a BFS rooted in v . For each node t extracted from the BFS queue, we update the distances between all affected sources s of t and t (Line 10) and enqueue the unvisited neighbors of t , if their distance from u has decreased as a consequence of the edge insertion (Line 15). When enqueueing a neighbor w of t , we set t to $P(w)$ (Line 18). This will be used to reduce the search space of affected sources in Line 9: only nodes that were affected sources of a predecessor will be examined. For example, if in Line 10 we find out that a node s is not an affected source for t , it will not be necessary to examine it again when the node extracted from the queue will be w (for the property represented in Figure 8).

QUINCA needs to store the distances d_{st} for each node pair (s, t) , which requires $\Theta(n^2)$ memory, and the number of affected sources of each affected target, which is $O(n)$ per node. The overall memory requirement of the algorithm is therefore $\Theta(n^2)$. Also its running time is $O(n^2)$. In fact, QUINCA basically runs two (truncated) BFSs to identify the affected targets and the affected sources, which requires $O(m)$ time, and then computes the distance between each affected target and the affected sources, resulting in $O(n^2)$ operations.

Since QUINCA was shown to be significantly faster than all existing algorithms for incremental APSP, we use it as a building block for our incremental algorithm for the betweenness centrality of a single node.

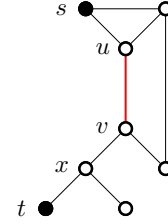


Fig. 8. Insertion of edge (u, v) creates a shortcut between s and t , but also between s and x and s and v .

6.2. New dynamic algorithm for the betweenness of a single node

Algorithm 18 describes the update of pairwise distances after a node insertion. For that, we only need to store the pairwise distances d_{st} for each $s, t \in V$. When computing the betweenness of a specific node x , we also need additional information. In particular, we need the number of shortest paths σ_{st} between each node pair (s, t) and the number σ_{stx} of shortest paths between s and t that go through x . Then, we can compute betweenness by using its definition given in Eq. (1). In the following, we will assume that the graph G is unweighted and connected, but the algorithm can be easily extended to weighted and disconnected graphs, in a way analogous to what has been done in [Slobbe et al. 2016]. Our algorithm can be divided in two phases: an *initialization* phase, where pairwise distances,

Algorithm: QUINCA
Input : Graph $G = (V, E)$, edge insertion (u, v) , pairwise distances $d_{(\cdot, \cdot)}$
Output : Updated pairwise distances
Assume: boolean $vis(v)$ is false, $\forall v \in V$

```

1  $S(v) \leftarrow \text{findAffectedSources}(G, (u, v), d)$ ;
2  $d_{uv} \leftarrow 1$ ;
3  $Q \leftarrow \emptyset$ ;
4  $P(v) \leftarrow v$ ;
5  $Q.\text{push}(v)$ ;
6  $vis(v) \leftarrow \text{true}$ ;
7 while  $Q.\text{length}() > 0$  do
8    $t = Q.\text{front}()$ ;
9   foreach  $s \in S(P(t))$  do */
10     if  $d_{st} > d_{su} + 1 + d_{vt}$  then
11        $d_{st} \leftarrow d_{su} + 1 + d_{vt}$ ;
12       if  $t \neq v$  then
13          $S(t).\text{insert}(s)$ ;
14   foreach  $w$  s.t.  $(t, w) \in E$  do */
15     if not  $vis(w)$  and  $d_{uw} > 1 + d_{vw}$  then
16        $Q.\text{push}(w)$ ;
17        $vis(w) \leftarrow \text{true}$ ;
18        $P(w) \leftarrow t$ ;
```

$\sigma_{(\cdot, \cdot)}$ and $\sigma_{(\cdot, \cdot)x}$ are computed and stored, and an *update* phase, where the data structures and the betweenness of node x are updated as a consequence of the edge insertion.

6.2.1. Initialization. The initialization can be easily done by running a Single-Source Shortest Path (SSSP) from each node, as in the first phase of Brandes's algorithm for betweenness centrality [Brandes 2001]. While computing distances from a source node s to any other node t , we set the number σ_{st} of shortest paths between s and t to the sum $\sum \sigma_{sp}$ over all nodes p such that $d_{st} = d_{sp} + 1$ (and we set $\sigma_{ss} = 1$). This can be done for a node s in $O(m)$ in unweighted graphs and in $O(m + n \log n)$ in weighted graphs (the cost of running a BFS or Dijkstra, respectively). Instead of discarding this information after each SSSP as in Brandes's algorithm, we store both the distances $d_{(\cdot, \cdot)}$ and the numbers of shortest paths $\sigma_{(\cdot, \cdot)}$ in a matrix. After this, we can compute the number $\sigma_{(\cdot, \cdot)x}$ of shortest paths going through x . For each node pair (s, t) , σ_{stx} is equal to $\sigma_{sx} \cdot \sigma_{xt}$ if $d_{st} = d_{sx} + d_{xt}$, and to 0 otherwise. The betweenness b_x of x can then be computed using the definition given in Eq. (1). This second part can be done in $O(n^2)$ time by looping over all node pairs. Therefore the total running time of the initialization is $O(nm)$ for unweighted graphs and $O(n(m + n \log n))$ for weighted graphs, and the memory requirement is $O(n^2)$, since we need to store three matrices of size $n \times n$ each.

6.2.2. Update. The update works in a way similar to QUINCA (Algorithm 18), with a few differences. First, in this case we cannot consider as affected only the nodes whose new distances d'_{st} is *strictly smaller* than the new distance d_{st} , but also the ones whose old distance d_{st} is *equal* to $d_{su} + 1 + d_{vt}$. In fact, the new insertion might create new shortest paths between two nodes, even when their distance does not change. Second, we need to compute the new σ'_{st} and σ'_{stx} for each affected node pair (s, t) . Algorithm 2 gives an overview of the algorithm for betweenness update for a single node x , whereas Algorithm 3 and Algorithm 4 describe the update $\sigma_{(\cdot, \cdot)}$ and $\sigma_{(\cdot, \cdot)x}$ when $d_{st} > d_{su} + 1 + d_{vt}$ and when $d_{st} = d_{su} + 1 + d_{vt}$, respectively.

Algorithm 2 shares its structure with QUINCA (Algorithm 18), so in the following we will only underline the differences between the two. In Line 1, `findAffectedSourcesGEQ` works like `findAffectedSources` of QUINCA, with the only difference that also nodes s such that $d_{sv} = d_{su} + 1$ are returned and not only those whose distance has decreased. Analogously, the symbol $>$ in Line 10 and Line 15 of QUINCA has been replaced with \geq in Line 15 and Line 28 in Algorithm 2.

In Lines 3-7, after setting the new distance between u and v , also σ_{uv} and σ_{uvx} are updated accordingly. Then (Lines 15-26), for each affected node pair (s, t) , we first subtract the old contribution σ_{stx}/σ_{st} from the betweenness of x , then we recompute d_{st} , σ_{st} and σ_{stx} with either `updateSigmaGR` or `updateSigmaEQ`, and finally we add the new contribution $\sigma'_{stx}/\sigma'_{st}$ to b_x . Notice that, if x did not lie in any shortest path between s and t before the edge insertion, $\sigma_{stx} = 0$ and therefore b_x is not decreased in Line 17. Analogously, if x is not part of a shortest path between s and t after the insertion, b_x is not increased in Line 24.

In the following, we analyze `updateSigmaGR` and `updateSigmaEQ` separately.

ALGORITHM 2: Update of b_x after an edge insertion

Algorithm: Incremental betweenness

Input : Graph $G = (V, E)$, edge insertion (u, v) , pairwise distances $d_{(\cdot, \cdot)}$, numbers $\sigma_{(\cdot, \cdot)}$ of shortest paths, numbers $\sigma_{(\cdot, \cdot)x}$ of shortest paths through x , betweenness value b_x of x

Output : Updated $d_{(\cdot, \cdot)}$, $\sigma_{(\cdot, \cdot)}$, $\sigma_{(\cdot, \cdot)x}$ and b_x

Assume: boolean $vis(v)$ is false, $\forall v \in V$

```

1  $S(v) \leftarrow \text{findAffectedSourcesGEQ}(G, (u, v), d);$ 
2  $d_{uv} \leftarrow 1;$ 
3  $\sigma_{uv} \leftarrow 1;$ 
4 if  $x = u$  or  $x = v$  then
5    $\sigma_{uvx} \leftarrow 1;$ 
6 else
7    $\sigma_{uvx} \leftarrow 0;$ 
8  $Q \leftarrow \emptyset;$ 
9  $P(v) \leftarrow v;$ 
10  $Q.\text{push}(v);$ 
11  $vis(v) \leftarrow \text{true};$ 
12 while  $Q.\text{length}() > 0$  do
13    $t = Q.\text{front}();$ 
14   foreach  $s \in S(P(t))$  do
15     if  $d_{st} \geq d_{su} + 1 + d_{vt}$  then
16       if  $x \neq s$  and  $x \neq t$  then
17          $b_x \leftarrow b_x - \sigma_{stx}/\sigma_{st};$ 
18       if  $d_{st} > d_{su} + 1 + d_{vt}$  then
19          $\sigma_{st}, \sigma_{stx} \leftarrow \text{updateSigmaGR}(G, (u, v), d, \sigma, \sigma_x);$ 
20          $d_{st} \leftarrow d_{su} + 1 + d_{vt};$ 
21       else
22          $\sigma_{st}, \sigma_{stx} \leftarrow \text{updateSigmaEQ}(G, (u, v), d, \sigma, \sigma_x);$ 
23       if  $x \neq s$  and  $x \neq t$  then
24          $b_x \leftarrow b_x + \sigma_{stx}/\sigma_{st};$ 
25       if  $t \neq v$  then
26          $S(t).\text{insert}(s);$ 
27   foreach  $w$  s.t.  $(t, w) \in E$  do
28     if not  $vis(w)$  and  $d_{uw} \geq 1 + d_{vw}$  then
29        $Q.\text{push}(w);$ 
30        $vis(w) \leftarrow \text{true};$ 
31        $P(w) \leftarrow t;$ 

```

UpdateSigmaGR. Let us consider the case $d_{st} > d_{su} + 1 + d_{vt}$. In this case, all the old shortest paths are discarded, as they are not shortest paths any longer, and all the new shortest paths go through edge (u, v) . Therefore, we can set the new number σ'_{st} of shortest paths between s and t to $\sigma_{su} \cdot \sigma_{vt}$. Since all old shortest paths should be discarded, also σ_{stx} depends only on the new shortest paths and not on whether x used to lie in some shortest paths between s and t before the edge insertion. Depending on the position of x with respect to the new shortest paths, we can define three cases, depicted in Figure 9. In Case (a) (left), x lies in one of the shortest paths between s and u . This means that it also lies in some shortest paths between s and t . In particular, the number of these paths σ'_{stx} is equal to $\sigma_{sux} \cdot \sigma_{vt}$. Notice that no shortest paths between s and u can be affected (see [Slobbe et al. 2016] for the proof) and therefore $\sigma_{sux} = \sigma'_x(s, u)$. In Case (b) (center), x lies in one of the shortest paths between v and t . Analogously to Case 1, the new number of shortest paths between s and t going through x is $\sigma'_{stx} = \sigma_{su} \cdot \sigma_{vtx}$. Notice that Case (a) and Case (b) cannot both be true at the same time. In fact, if $d_{su} = d_{sx} + d_{xu}$ and $d_{vt} = d_{vx} + d_{xt}$, we would have that $d'_{st} = d_{su} + 1 + d_{vt} = d_{sx} + d_{xu} + 1 + d_{vx} + d_{xt} > d_{sx} + d_{xt}$, which is impossible, since d'_{st} is the shortest-path distance between s and t . Therefore, at least one among σ_{sux} and σ_{vtx} must be equal to 0. Finally, in Case (c) (right), σ_{sux} and σ_{vtx} are both equal to 0, meaning that x does not lie on any new shortest path between s and t . Once again, this is independent on whether x lied in an old shortest path between s and t or not. Algorithm 3 shows the computation of σ'_{st} and σ'_{stx} . Notice that, in the computation of σ'_{stx} , the first addend is greater than zero only in Case (a) and the second only in Case (b).

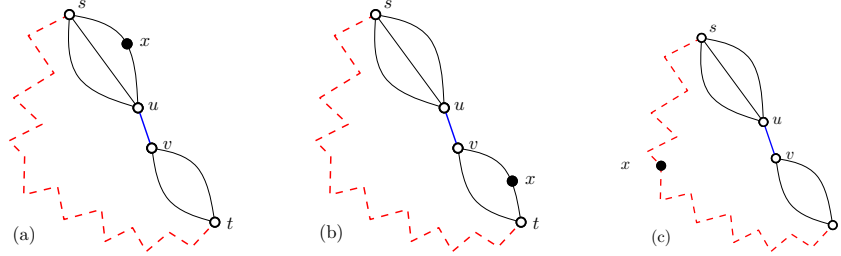


Fig. 9. Possible positions of x with respect to the new shortest paths after the insertion of edge (u, v) . On the left, x lies between the source s and u . In the center, x lies between v and the target t . On the right, x does not lie on any new shortest path between s and t .

ALGORITHM 3: Update of σ_{st} and σ_{stx} when (u, v) creates new shortest paths of length smaller than d_{st}

Algorithm: UpdateSigmaGR

Input : Graph $G = (V, E)$, edge insertion (u, v) , pairwise distances $d_{(\cdot, \cdot)}$, numbers $\sigma_{(\cdot, \cdot)}$ of shortest paths, numbers $\sigma_{(\cdot, \cdot)x}$ of shortest paths through x

Output : Updated σ'_{st} , σ'_{stx}

- 1 $\sigma'_{st} \leftarrow \sigma_{su} \cdot \sigma_{vt}$;
 - 2 $\sigma'_{stx} \leftarrow \sigma_{sux} \cdot \sigma_{vt} + \sigma_{su} \cdot \sigma_{vtx}$;
 - 3 **return** σ'_{st} , σ'_{stx} ;
-

UpdateSigmaEQ. Let us now consider the case $d_{st} = d_{su} + 1 + d_{vt}$. Here all the old shortest paths between s and t are still valid and, in addition to them, new shortest paths going through (u, v) have been created. Therefore, the new number of shortest paths σ'_{st} is simply $\sigma_{st} + \sigma_{su} \cdot \sigma_{vt}$. Notice that we never count the same path multiple times, since all new paths go through (u, v) and none of the old paths does. Also all old shortest paths between s and t through x are still valid, therefore σ'_{stx} is given by the old σ_{stx} plus the number of new shortest paths going through both x and (u, v) . This number can be computed as described for **updateSigmaGR** according to the cases of Figure 9. Algorithm 4 shows the computation of σ'_{st} and σ'_{stx} .

ALGORITHM 4: Update of σ_{st} and σ_{stx} when (u, v) creates new shortest paths of length equal to d_{st}

Algorithm: UpdateSigmaEQ

Input : Graph $G = (V, E)$, edge insertion (u, v) , pairwise distances $d_{(\cdot, \cdot)}$, numbers $\sigma_{(\cdot, \cdot)}$ of shortest paths, numbers $\sigma_{(\cdot, \cdot)x}$ of shortest paths through x

Output : Updated $\sigma'_{st}, \sigma'_{stx}$

```

1  $\sigma'_{st} \leftarrow \sigma_{st} + \sigma_{su} \cdot \sigma_{vt};$ 
2  $\sigma'_{stx} \leftarrow \sigma_{stx} + \sigma_{sux} \cdot \sigma_{vt} + \sigma_{su} \cdot \sigma_{vtx};$ 
3 return  $\sigma'_{st}, \sigma'_{stx};$ 

```

6.3. Time complexities

6.3.1. Dynamic betweenness algorithm.. When describing the dynamic APSP algorithm in Section 6.1, we defined the set of affected targets of a node $s \in V$ as $\{t \in V : d'_{st} < d_{st}\}$ (and, analogously, the set of affected sources of t as $\{s \in V : d'_{st} < d_{st}\}$). As explained in Section 6.2, in the betweenness update problem we need to consider as affected all the node pairs that have shortest paths going through the inserted edge, even if their distance has not decreased. Therefore, we redefine the set $T(s)$ of affected targets of a node $s \in V$ as $\{t \in V : d_{su} + 1 + d_{vt} \leq d_{st}\}$ and the set $S(t)$ of affected sources of a node $t \in V$ as $\{s \in V : d_{su} + 1 + d_{vt} \leq d_{st}\}$. Let us define the *extended size* $\|A\|$ of a set of nodes A as the sum of the number of nodes in A and the number of edges that have a node of A as their endpoint. Then, the following proposition holds.

PROPOSITION 6.1. *The running time of Algorithm 2 for updating the betweenness of a single node after an edge insertion (u, v) is $\Theta(\|S(v)\| + \|T(u)\| + \sum_{y \in T(u)} |S(P(y))|)$.*

PROOF. The function **findAffectedSourcesGEQ** in Line 1 identifies the set of affected sources starting a BFS in v and visiting only the nodes s such that $d_{su} + 1 + d_{vt} \leq d_{st}$. This takes $\Theta(\|S(v)\|)$, since this partial BFS visits all nodes in $S(v)$ and their incident edges. Then, the while loop of Lines 12 - 31 (excluding the part in Lines 14 - 26) identifies all the affected targets $T(u)$ with a partial BFS. This part requires $\Theta(\|T(u)\|)$ operations, since all affected targets and their incident edges are visited. In Lines 14 - 26, for each affected node $t \in T(u)$, all the affected sources of the predecessor $P(t)$ of t are scanned. This part requires in total $\Theta(\sum_{t \in T(u)} |S(P(t))|)$ operations, since for each node in $S(P(t))$, Lines 15 - 26 require constant time. \square

Notice that, since $S(P(y))$ is $O(n)$ and both $\|T(u)\|$ and $\|S(v)\|$ are $O(n + m)$, the worst-case complexity of Algorithm 2 is $O(n^2)$ (assuming $m = \Omega(n)$). This matches the worst-case running time of the dynamic APSP algorithm **QUINCA**. Also, this introduces a contrast between the static and the incremental case: Whereas the static computation of one node's betweenness has the same complexity as computing it for all nodes (at least no algorithm for computing it for one node faster than computing it for all nodes exists so far), in the

incremental case the betweenness update of a single node can be done in $O(n^2)$, whereas there is no algorithm faster than $O(nm)$ for the update of all nodes.

6.3.2. Greedy algorithm for betweenness maximization.. We can improve the running time of Algorithm GREEDY by using the dynamic algorithm for betweenness centrality instead of the recomputation from scratch. In fact, at Line 4 of Algorithm GREEDY, we add an edge $\{u, v\}$ to the graph and compute the betweenness in the new graph, for each node u in $V \setminus N_v(S)$. If we compute the betweenness by using the from-scratch algorithm, this step requires $O(nm)$ and this leads to an overall complexity of $O(kn^2m)$. At Line 4, instead of recomputing betweenness on the new graph from scratch, we can use Algorithm 2. As we proved previously, its worst-case complexity is $O(n^2)$. This leads to an overall worst-case complexity of $O(kn^3)$ for GREEDY. However, in Section 6.4 we will show that GREEDY is actually much faster in practice.

6.4. Experimental evaluation

In this section we evaluate the running time of the dynamic algorithm for betweenness centrality computation. We used the same experimental setting used in Section 5.1. For our experiments, we consider a set of real-world networks belonging to different domains, taken from SNAP [Leskovec and Krevl 2014], KONECT [Kunegis 2013], Pajek [Batagelj and Mrvar 2006], and the 10th DIMACS Implementation Challenge [Bader et al. 2014]. The properties of the networks are reported in Table II (directed graphs) and in Table III (undirected graphs). Since some of the algorithms we use for comparison work only on unweighted graphs, all the tested networks are unweighted (although we recall that our algorithm described in Section 6.2 can handle also weighted graphs).

6.4.1. Evaluation of the dynamic algorithm for the betweenness of one node.. In the following, we refer to our incremental algorithm for the update of the betweenness of a single node as **SI** (Single-node Incremental). Since there are no other algorithms specifically designed to compute or update the betweenness of a single node, we also use the static algorithm by Brandes [Brandes 2001] and the dynamic algorithm by Green et al. [Green et al. 2012] for a comparison (the one by Brandes was already in NetworKit). Indeed, the algorithm by Brandes (which we refer to as **Stat**, from Static) is the best known algorithm for static computation of betweenness and the one by Green et al. (which we name **AI**, from All-nodes Incremental) is the dynamic algorithm for which the best speedups over Brandes’s algorithm are reported [Bergamini et al. 2015; Green et al. 2012].

To compare the running times of the algorithms for betweenness centrality, we choose a node x at random and we assume we want to compute the betweenness of x . Then, we add an edge to the graph, also chosen uniformly at random among the node pairs (u, v) such that $(u, v) \notin E$. After the insertion, we use the three algorithms to update the betweenness centrality of x and compare their running times. We recall that **Stat** is a static algorithm, which means that we can only run it from scratch on the graph after the edge insertion. On each graph, we repeat this 20 times and report the average running time obtained by each of the four algorithms. Table II and Table III show the results for directed and undirected graphs, respectively. As expected, both dynamic algorithms **AI** and **SI** are faster than the static approach and **SI** is the fastest among all algorithms, since it is the one that performs the smallest number of operations.

The last two columns of Table II and Table III show the speedups of **SI** on **AI** and of **SI** on **Stat**, respectively. **SI** is always at least 20 times faster than **AI** on directed graphs and 82 on undirected, on average by a factor 91 for directed and a factor 398 for undirected graphs (geometric means). These high speedups are mainly due to two factors: the first is that, when focusing on a single node, we do not need to update the scores of all the nodes that lie in some shortest path affected by the edge insertion. The second reason is due to the way **AI** recomputes the pairwise distances. Indeed, let us assume that edge (u, v)

Table II. Running times of the betweenness algorithms on directed real-world graphs. The last two columns report the speedups of our incremental algorithm (SI) on the incremental algorithm for the betweenness of all nodes (AI) and on the static algorithm (Stat), respectively.

Graph	Nodes	Edges	Time Stat [s]	Time AI [s]	Time SI [s]	Sp. SI on AI	Sp. SI on Stat
subelj-jung	6 120	50 535	5.51711	0.04700	0.00076	62.08	7286.14
wiki-Vote	7 115	100 762	32.53902	0.55123	0.01062	51.88	3062.62
elec	7 118	103 617	36.06548	1.36465	0.01791	76.21	2014.16
freeassoc	10 617	63 788	108.78891	2.05435	0.02213	92.83	4915.71
dblp-cite	12 591	49 728	37.41786	0.39830	0.00519	76.73	7207.86
subelj-cora	23 166	91 500	51.49332	1.05313	0.05211	20.21	988.20
ego-twitter	23 370	33 101	11.73085	0.00685	0.00008	90.33	154701.31
ego-gplus	23 628	39 242	13.85485	0.00901	0.00009	102.10	157057.98
munmun-digg	30 398	85 247	133.09115	2.53912	0.00901	281.89	14775.73
linux	30 837	213 424	45.18776	0.39286	0.00091	429.88	49445.29

Table III. Running times of the betweenness algorithms on undirected real-world graphs. The last two columns report the speedups of our incremental algorithm (SI) on the incremental algorithm for the betweenness of all nodes (AI) and on the static algorithm (Stat), respectively.

Graph	Nodes	Edges	Time Stat [s]	Time AI [s]	Time SI [s]	Sp. SI on AI	Sp. SI on Stat
Mus-musculus	4 610	5 747	5.27364	0.29210	0.00262	111.41	2011.40
HC-BIOGRID	4 039	10 321	9.07513	0.51905	0.00630	82.33	1439.54
Caenor-eleg	4 723	9 842	8.94099	0.60436	0.00254	238.16	3523.36
ca-GrQc	5 241	14 484	7.57090	0.56623	0.00261	216.84	2899.33
advogato	7 418	42 892	20.74108	0.99484	0.00209	475.00	9903.17
hprd-pp	9 465	37 039	50.99403	3.48279	0.00402	866.08	12680.91
ca-HepTh	9 877	25 973	39.86048	2.54398	0.00685	371.18	5815.79
dr-melanog	10 625	40 781	68.74479	3.23557	0.00647	499.72	10617.35
oregon1	11 174	23 409	57.87351	2.33477	0.00400	583.57	14465.29
oregon2	11 461	32 730	65.94178	2.66197	0.00371	717.95	17784.83
Homo-sapiens	13 690	61 130	124.85679	8.31994	0.00727	1145.20	17185.88
GoogleNw	15 763	148 585	145.38185	1.95749	0.00716	273.29	20297.03
CA-CondMat	21 363	91 342	310.47983	22.74763	0.01703	1335.55	18228.79

has been inserted. For each affected source node s (i.e. nodes for which $d_{sv} \geq d_{su} + 1$), AI runs a (partial) SSSP from v to identify the affected target nodes (i.e., nodes t for which $d_{st} \geq d_{su} + 1 + d_{vt}$). This implies that some edges are visited by AI multiple times. For example, the outgoing edges of v are visited once for each affected source node. On the contrary, SI updates the distances based on QUINCA, which runs only one (partial) SSSP from u and one from v and then compares the distances of the affected nodes and the affected targets. The combination of these two factors make SI extremely fast: on all tested instances, its running time is always smaller than 0.1 seconds, whereas AI can take up to several seconds to recompute betweenness.

Compared to recomputation, SI is on average about 11000 times faster than Stat on directed and about 7800 times on undirected graphs (geometric means of the speedups). Since SI has shown to outperform other approaches in the context of updating the betweenness centrality of a single node after an edge insertion, we use it to update the betweenness in the greedy algorithm for the Maximum Betweenness Improvement problem (Section 5). Therefore, in all the following experiments, what we refer to as GREEDY is the Algorithm of Section 5 where we recompute betweenness after each edge insertion with SI.

6.4.2. Running times of the greedy algorithm for betweenness maximization.. In Section 5, we already showed that GREEDY outperforms all other heuristics in terms of solution quality, both on directed and on undirected graphs (although we recall that the theoretical guarantee on the approximation ratio holds only for directed graphs). In this section, we report the running times of GREEDY, using SI to recompute betweenness. Table IV and Table V show the results on directed and undirected graphs, respectively. For each value of k , the tables show the running time required by GREEDY when k edges are added to the graph. Notice that this is not the running time of the k th iteration, but the total running time of GREEDY for a certain value of k . Since on directed graphs the betweenness of x is a submodular

ALGORITHM 5: GREEDY algorithm with pruning (exploiting submodularity).

Input : A directed graph $G = (V, E)$; a vertex $v \in V$; and an integer $k \in \mathbb{N}$

Output: Set of edges $S \subseteq \{(u, v) \mid u \in V \setminus N_v\}$ such that $|S| \leq k$

```

1  $S \leftarrow \emptyset$ ;
2 foreach  $u \in V \setminus (N_v(S))$  do
3    $\Delta b_v(u) \leftarrow 0$ 
4 for  $i = 1, 2, \dots, k$  do
5    $LB \leftarrow 0$ ;
6   foreach  $u \in V \setminus (N_v(S))$  do
7     if  $(i = 1) \vee (LB < (b_v(S) + \Delta b_v(u)))$  then
8       Compute  $b_v(S \cup \{(u, v)\})$ 
9        $\Delta b_v(u) \leftarrow b_v(S \cup \{(u, v)\}) - b_v(S)$ ;
10       $LB \leftarrow \max(LB, b_v(S \cup \{(u, v)\}))$ ;
11    $u_{\max} \leftarrow \arg \max \{b_v(S \cup \{(u, v)\}) \mid u \in V \setminus (N_v(S))\}$ ;
12    $S \leftarrow S \cup \{(u_{\max}, v)\}$ ;
13 return  $S$ ;

```

function of the solutions for MBI (see Theorem 5.2), we can speed up the computation for $k > 1$ (see Algorithm 5). Let $\Delta b_v(u) = b_v(S \cup \{(u, v)\}) - b_v(S)$, where S is the solution computed at some iteration $i' < i$, that is, $\Delta b_v(u)$ is the increment to b_v given by adding the edge (u, v) to S at iteration i' . Let LB be the current best solution at iteration i . We avoid to compute $b_v(S \cup \{(u, v)\})$ at line 8 if $LB \geq b_v(S) + \Delta b_v(u)$. In fact, by definition of submodularity, $\Delta b_v(u)$ is monotonically non-increasing and $b_v(S) + \Delta b_v(u)$ is an upper bound for $b_v(S \cup \{(u, v)\})$. Then, $LB \geq b_v(S) + \Delta b_v(u)$ implies $LB \geq b_v(S \cup \{(u, v)\})$.

Our experiments show that exploiting submodularity has significant effects on the running times: for all graphs in Table IV, we see that the difference in running time between computing the solution for $k = 1$ and $k = 10$ is at most a few seconds. Also, for all graphs the computation never takes more than a few minutes.

Table IV. Running times (in seconds) of GREEDY on directed real-world graphs for different values of k .

Graph	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$	$k = 9$	$k = 10$
subelj-jung	6.53	8.33	8.51	8.68	8.84	9.00	9.16	9.32	9.49	9.64
wiki-Vote	8.35	8.40	8.44	8.47	8.51	8.54	8.57	8.60	8.62	8.64
elec	12.99	13.03	13.06	13.09	13.12	13.16	13.20	13.23	13.26	13.28
freeassoc	112.23	116.93	121.85	127.05	129.67	132.10	134.19	136.08	138.92	140.14
subelj-cora	289.19	307.48	320.42	326.28	331.63	335.11	338.44	341.71	344.94	348.16
ego-twitter	5.21	5.23	5.33	5.54	5.59	5.64	5.69	5.73	5.76	5.79
ego-gplus	41.86	42.04	42.30	42.41	43.19	44.04	44.12	44.38	44.47	45.20
munmun-digg	729.64	730.01	730.49	730.72	730.99	731.19	731.42	732.27	732.50	732.67
linux	428.32	446.03	447.43	448.39	449.52	450.51	451.61	452.32	452.68	453.36

Unfortunately, submodularity does not hold for undirected graphs, therefore for each k we need to apply SI to all possible new edges between x and other nodes. Nevertheless, apart from the CA-CondMat graph (where it takes about 8 hours for $k = 10$), GREEDY never requires more than 1 hour for $k = 10$. For $k = 1$, it takes at most a few minutes. Quite surprisingly, the running time of the first iteration is often smaller than that of the following ones, in particular if we consider that the first iteration also includes the initialization of SI. This might be due to the fact that, initially, the pivots are not very central and therefore many edge insertions between the pivots and other nodes affect only a few shortest paths. Since the running time of IA is proportional to the number of affected node pairs, this makes it very fast during the first iteration. On the other hand, at each iteration the pivot x gets more and more central, affecting a greater number of nodes when a new shortcut going through x is created. It is also interesting to notice that, on several graphs, the first

Table V. Running times (in seconds) of GREEDY on undirected real-world graphs for different values of k .

Graph	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$	$k = 9$	$k = 10$
Mus-musculus	13.4	68.8	151.5	245.3	345.4	391.9	434.7	480.7	527.8	575.3
HC-BIOGRID	15.4	65.1	137.8	233.5	349.0	434.9	517.1	595.0	676.3	760.0
Caenor-eleg	4.7	21.8	53.0	96.8	150.5	213.8	287.5	369.3	459.4	557.3
ca-GrQc	7.1	20.3	42.0	70.8	106.5	149.5	201.4	232.4	270.0	310.8
advogato	7.0	14.1	24.7	39.6	58.1	79.2	102.9	129.4	158.6	190.7
hprd-pp	38.0	86.8	154.9	238.7	345.1	468.9	607.1	758.7	924.1	1097.7
ca-HepTh	90.8	363.0	759.3	1201.9	1696.6	1890.7	2116.0	2367.2	2641.4	2933.3
dr-melanog	45.1	105.4	192.7	302.1	432.8	584.2	757.6	952.9	1170.6	1410.2
oregon1	32.4	92.9	165.7	216.6	276.5	345.4	420.6	477.8	535.5	596.7
oregon2	18.5	41.7	86.8	116.3	157.9	212.7	258.7	310.0	368.1	430.2
Homo-sapiens	76.2	223.1	415.5	663.9	950.0	1271.9	1628.1	1820.1	2028.8	2259.6
GoogleNw	83.9	152.2	225.4	303.9	373.2	445.0	519.5	574.5	632.2	691.9
CA-CondMat	831.0	2899	5868	8806	11806	14749	17816	21197	24879	28832

iteration of GREEDY is even faster than computing the betweenness once with Brandes’s algorithm (see Table II and Table III for comparison). This is due to the fact that the initialization of SI described in Section 6.2 is often much faster than Brandes’s algorithm. Indeed, it only computes the shortest paths between the node pairs and “skips” the second part of Brandes’s algorithm where the betweenness scores are computed.

To summarize, our experimental results show that our incremental algorithm for the betweenness of one node is much faster than existing incremental algorithms for the betweenness of all nodes, taking always fractions of seconds even when the competitor takes several seconds. The combination of it with our greedy approach for the MBI problem allows us to maximize betweenness of graphs with hundreds of thousands of edges in reasonable time. Also, our results in Section 5.1 show that GREEDY outperforms other heuristics both on directed and undirected graphs and both for the problem of betweenness and ranking maximization.

7. CONCLUSIONS

Betweenness centrality is a widely-used metric that ranks the importance of nodes in a network. Since in several scenarios a high centrality directly translates to some profit, in this paper we have studied the problem of maximizing the betweenness of a vertex by inserting a predetermined number of new edges incident to it. Our greedy algorithm, which is a $(1 - \frac{1}{e})$ -approximation of the optimum for directed graphs, yields betweenness scores that are significantly higher than several other heuristics, both on directed and undirected graphs. Our results are drawn from experiments on a diverse set of real-world directed and undirected networks with up to 10^5 edges.

Also, combining our greedy approach with a new incremental algorithm for recomputing the betweenness of a node after an edge insertion, we are often able to find a solution in a matter of seconds or few minutes. Our new incremental algorithm extends a recently published APSP algorithm and is the first to recompute the betweenness of one node in $O(n^2)$ time. All other existing approaches recompute the betweenness of all nodes and require at least $O(nm)$ time, matching the worst-case complexity of the static algorithm. Although extremely fast, our betweenness update algorithm has a memory footprint of $\Theta(n^2)$, which is a limitation for very large networks. A possible direction for future work could be to combine our greedy approach with dynamic algorithms that compute an approximation of betweenness centrality. Since these algorithms require less memory than the exact ones, they might allow us to target even larger networks.

Also, future work could consider extensions of the problem studied in this paper, such as allowing additions of edges incident to other vertices or weight changes to the existing edges.

ACKNOWLEDGMENTS

This work is partially supported by German Research Foundation (DFG) grant ME-3619/3-1 (FINCA) within the Priority Programme 1736 *Algorithms for Big Data*. We thank Dominik Kiefer (Karlsruhe Institute of Technology) for help with the experimental evaluation.

REFERENCES

- Konstantin Avrachenkov and Nelly Litvak. 2006. The Effect of New Links on Google Pagerank. *Stoc. Models* 22, 2 (2006), 319–331.
- David A. Bader, Henning Meyerhenke, Peter Sanders, Christian Schulz, Andrea Kappes, and Dorothea Wagner. 2014. Benchmarking for Graph Clustering and Partitioning. In *Encyclopedia of Social Network Analysis and Mining*. Springer, 73–82.
- Vladimir Batagelj and Andrej Mrvar. 2006. Pajek datasets. (2006). <http://vlado.fmf.uni-lj.si/pub/networks/data/>
- Reinhard Bauer, Gianlorenzo D’Angelo, Daniel Delling, Andrea Schumm, and Dorothea Wagner. 2012. The Shortcut Problem - Complexity and Algorithms. *J. Graph Algorithms Appl.* 16, 2 (2012), 447–481.
- Pietro Belotti, Jon Lee, Leo Liberti, François Margot, and Andreas Wächter. 2009. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software* 24, 4-5 (2009), 597–634.
- Elisabetta Bergamini, Michele Borassi, Pierluigi Crescenzi, Andrea Marino, and Henning Meyerhenke. 2016. Computing Top- k Closeness Centrality Faster in Unweighted Graphs. In *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016*. 68–80.
- Elisabetta Bergamini and Henning Meyerhenke. 2016. Approximating Betweenness Centrality in Fully Dynamic Networks. *Internet Mathematics* 12, 5 (2016), 281–314.
- Elisabetta Bergamini, Henning Meyerhenke, and Christian Staudt. 2015. Approximating Betweenness Centrality in Large Evolving Networks. In *Proceedings of the Seventeenth Workshop on Algorithm Engineering and Experiments, ALENEX 2015*, Ulrik Brandes and David Eppstein (Eds.). SIAM, 133–146.
- Davide Bilò, Luciano Gualà, and Guido Proietti. 2012. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theor. Comput. Sci.* 417 (2012), 12–22.
- Béla Bollobás, Christian Borgs, Jennifer Chayes, and Oliver Riordan. 2003. Directed scale-free graphs. In *Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms (SODA)*. SIAM, 132–139.
- Michele Borassi and Emanuele Natale. 2016. KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation. *CoRR* abs/1604.08553 (2016).
- Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology* 25 (2001), 163–177.
- Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Alessandro Panconesi, and Prabhakar Raghavan. 2009. Models for the compressible web. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 331–340.
- Pierluigi Crescenzi, Gianlorenzo D’Angelo, Lorenzo Severini, and Yllka Velaj. 2015. Greedily improving our own centrality in a network. In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA 2015) (Lecture Notes in Computer Science)*, Vol. 9125. Springer, 43–55.
- Pierluigi Crescenzi, Gianlorenzo D’Angelo, Lorenzo Severini, and Yllka Velaj. 2016. Greedily Improving Our Own Closeness Centrality in a Network. *ACM Transactions on Knowledge Discovery from Data* 11, 1, Article 9 (2016), 32 pages.
- Gianlorenzo D’Angelo, Lorenzo Severini, and Yllka Velaj. 2016. On the maximum betweenness improvement problem. In *Proceedings of the 16th Italian Conference on Theoretical Computer Science (ICTCS15) (Electr. Notes Theor. Comput. Sci.)*, Vol. 322. 153–168.
- Sina Dehghani, Mohammad Amin Fazli, Jafar Habibi, and Sadra Yazdanbod. 2015. Using Shortcut Edges to Maximize the Number of Triangles in Graphs. *Operations Research Letters* 43, 6 (2015), 6.
- Erik D. Demaine and Morteza Zadimoghaddam. 2010. Minimizing the Diameter of a Network Using Shortcut Edges. In *Proc. of the 12th Scandinavian Symp. and Work. on Algorithm Theory (SWAT) (Lecture Notes in Computer Science)*, Vol. 6139. Springer, 420–431.
- Camil Demetrescu and Giuseppe F. Italiano. 2004. A new approach to dynamic all pairs shortest paths. *J. ACM* 51, 6 (2004), 968–992.
- Martin G. Everett and Thomas W. Valente. 2016. Bridging, brokerage and betweenness. *Social Networks* 44 (2016), 202–208.
- Uriel Feige. 1998. A Threshold of $\ln N$ for Approximating Set Cover. *J. ACM* 45, 4 (1998).

- Fabrizio Frati, Serge Gaspers, Joachim Gudmundsson, and Luke Mathieson. 2015. Augmenting Graphs to Minimize the Diameter. *Algorithmica* 72, 4 (2015), 995–1010.
- Linton C. Freeman. 1977. A Set of Measures of Centrality Based on Betweenness. *Sociometry* 40, 1 (1977), 35–41.
- M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York.
- Robert Geisberger, Peter Sanders, and Dominik Schultes. 2008. Better Approximation of Betweenness Centrality. In *Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments, ALENEX 2008, San Francisco, California, USA, January 19, 2008*, J. Ian Munro and Dorothea Wagner (Eds.). SIAM, 90–100.
- Oded Green, Robert McColl, and David A. Bader. 2012. A Fast Algorithm for Streaming Betweenness Centrality. In *SocialCom/PASSAT*. IEEE, 11–20.
- Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. 2015. Fully Dynamic Betweenness Centrality Maintenance on Massive Networks. *PVLDB* 9, 2 (2015), 48–59.
- Vatche Ishakian, Dóra Erdős, Evimaria Terzi, and Azer Bestavros. 2012. A Framework for the Evaluation and Management of Network Centrality. In *Proc. of the 12th SIAM Int. Conf. on Data Mining (SDM)*. SIAM, 427–438.
- Miray Kas, Matthew Wachs, Kathleen M. Carley, and L. Richard Carley. 2013. Incremental algorithm for updating betweenness centrality in dynamically growing networks. In *Advances in Social Networks Analysis and Mining 2013 (ASONAM '13)*. ACM, 33–40.
- N. Kourtellis, G. De Francisci Morales, and F. Bonchi. 2015. Scalable Online Betweenness Centrality in Evolving Graphs. *Knowledge and Data Engineering, IEEE Transactions on PP*, 99 (2015), 1–1.
- Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D Sivakumar, Andrew Tomkins, and Eli Upfal. 2000. Stochastic models for the web graph. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 57–65.
- Jérôme Kunegis. 2013. KONECT: the Koblenz network collection. In *22nd Int. World Wide Web Conf., WWW '13*. 1343–1350. <http://dl.acm.org/citation.cfm?id=2488173>
- Min-Joong Lee, Sunghye Choi, and Chin-Wan Chung. 2016. Efficient algorithms for updating betweenness centrality in fully dynamic graphs. *Inf. Sci.* 326 (2016), 278–296.
- Min-Joong Lee and Chin-Wan Chung. 2014. Finding k-highest betweenness centrality vertices in graphs. In *23rd International World Wide Web Conference, WWW 2014*, Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel (Eds.). ACM, 339–340.
- Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. <http://snap.stanford.edu/data>. (June 2014).
- Rong-Hua Li and Jeffrey Xu Yu. 2015. Triangle minimization in large networks. *Knowledge and Information Systems* 45, 3 (2015), 617–643.
- Paolo Malighetti, Gianmaria Martini, Stefano Paleari, and Renato Redondi. 2009. *The Impacts of Airport Centrality in the EU Network and Inter-Airport Competition on Airport Efficiency*. Technical Report MPRA-7673. <http://econpapers.repec.org/paper/pramprapa/17673.htm>
- Adam Meyerson and Brian Tagiku. 2009. Minimizing Average Shortest Path Distances via Shortcut Edge Addition. In *Proc. of the 13th Int. Work. on Approx. Alg. for Comb. Opt. Prob. (APPROX) (Lecture Notes in Computer Science)*, Vol. 5687. Springer, 272–285.
- Meghana Nasre, Matteo Pontecorvi, and Vijaya Ramachandran. 2014. Betweenness Centrality - Incremental and Faster. In *Mathematical Foundations of Computer Science 2014 - 39th Int. Symp., MFCS 2014 (Lecture Notes in Computer Science)*, Vol. 8635. Springer, 577–588.
- G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming* 14, 1 (1978), 265–294.
- Martin Olsen and Anastasios Viglas. 2014. On the approximability of the link building problem. *Theor. Comput. Sci.* 518 (2014), 96–116.
- Manos Papagelis. 2015. Refining Social Graph Connectivity via Shortcut Edge Addition. *ACM Trans. Knowl. Discov. Data* 10, 2 (2015), 12.
- Manos Papagelis, Francesco Bonchi, and Aristides Gionis. 2011. Suggesting Ghost Edges for a Smaller World. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM '11)*. ACM, 2305–2308.
- Nikos Parotsidis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2015. Selecting Shortcuts for a Smaller World. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 28–36.
- S. Perumal, P. Basu, and Ziyu Guan. 2013. Minimizing Eccentricity in Composite Networks via Constrained Edge Additions. In *Military Communications Conference, MILCOM 2013 - 2013 IEEE*. 1894–1899.

- Matteo Pontecorvi and Vijaya Ramachandran. 2015. Fully Dynamic Betweenness Centrality. In *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings (Lecture Notes in Computer Science)*, Khaled M. Elbassioni and Kazuhisa Makino (Eds.), Vol. 9472. Springer, 331–342.
- G. Ramalingam and Thomas W. Reps. 1996. On the Computational Complexity of Dynamic Graph Problems. *Theor. Comput. Sci.* 158, 1&2 (1996), 233–277.
- Matteo Riondato and Evgenios M. Kornaropoulos. 2016. Fast approximation of betweenness centrality through sampling. *Data Min. Knowl. Discov.* 30, 2 (2016), 438–475.
- Matteo Riondato and Eli Upfal. 2016. To appear. ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages. In *22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2016)*.
- Sudip Saha, Abhijin Adiga, B. Aditya Prakash, and Anil Kumar S. Vullikanti. 2015. Approximation Algorithms for Reducing the Spectral Radius to Control Epidemic Spread. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 568–576.
- Arie Slobbe, Elisabetta Bergamini, and Henning Meyerhenke. 2016. Faster Incremental All-pairs Shortest Paths. *Karlsruhe Reports in Informatics* (2016). <http://digbib.ubka.uni-karlsruhe.de/volltexte/1000052742>
- Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. To appear. NetworKit: A Tool Suite for High-Performance Network Analysis. *Network Science* (To appear).
- Mikkel Thorup. 2004. Fully-Dynamic All-Pairs Shortest Paths: Faster and Allowing Negative Cycles. In *9th Scandinavian Workshop on Algorithm Theory, SWAT 2004 (LNCS)*, Vol. 3111. Springer, 384–396.
- Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. 2012. Gelling, and Melting, Large Graphs by Edge Manipulation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM '12)*. ACM, 245–254.
- D.P. Williamson and D.B. Shmoys. 2011. *The Design of Approximation Algorithms*. Cambridge University Press.
- Baoning Wu and Brian D. Davison. 2005. Identifying link farm spam pages. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005*, Allan Ellis and Tatsuya Hagino (Eds.). ACM, 820–829.